HIGH-SPEED, LOW COST TEST PLATFORM USING FPGA TECHNOLOGY

A Dissertation Thesis
Presented to
The Academic Faculty

By

Te-Hui Chen

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering

Georgia Institute of Technology
December 2016

HIGH-SPEED, LOW COST TEST PLATFORM USING FPGA TECHNOLOGY

Approved by:

Dr. David C Keezer, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Linda S Milor
School Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Abhijit Chatterjee
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Paul Kohl
School of Chemical and Biomolecular Engineering
Georgia Institute of Technology

Dr. Gee-Kung Chang
School of Electrical and Computer Engineering
Georgia Institute of Technology

Date Approved: August 16th, 2016

ACKNOWLEDGEMENTS

My career in Georgia Tech started in 2009. After five years undergraduate study and one year military service in Taiwan, I was admitted to pursue graduated degree in this great institution. It was the first time I am so far away from my home town and come to an unfamiliar country without others accompany, everything happened here seemed to be so uncertain to me. However, I was so lucky to have those wonderful people's guidance, support and great friendship to help me pass all the difficulties.

First and foremost, I would like to express my sincere gratitude to my advisors, Dr. David Keezer, for his continuous support throughout my graduate studies in Georgia Tech. His audience and wisdom have proven to be invaluable in the development of my career. Without his encouragement and patience, I would not be able to achieve all these works. I also want to thank my lab senior Carl Grey who had leaded me and provided all his technical expertise in the very beginning of my career. Furthermore, I would like to thank current lab mate Jiangchi Yang for keeping inspiring me new ideas in this area.

Thank you to my reading committee member committee members Dr. Abhijeet Chatterjee and Dr. Linda Milor, who have been highly supportive for this research and available for guidance and advice. Dr. Chatterjee has helped our group with project support, numerous publication and industry liaisons, and I would like to give a special thanks to him here. Furthermore I would like to thank Dr. Gee-Kung Chang and Dr. Paul Kohl for serving my dissertation defense committees. Also I want to express my gratitude Samsung Corp., who gave a great sponsorship to the two major projects of this research. With the help of Samsung and Dr. Hyun Choi, I was able to start my research and avoid too many mistakes.

I would like to acknowledge my intern advisor Mark Chen in Intel Corp. He have given me so much experience and knowledge of digital testing in industry, and also the

other specialized field such as firmware and optical testing which fulfilled my knowledge in these areas. Also I want to give a great thanks to Jeff Galloway, Blake Grey, Sen-Wen Hsiao in Silicon Creations, without their help, I would not have chance to finish my Ph.D. degree. Furthermore, I want to honor those people I have met in my graduate career, including the classmates, course professors and all Taiwanese friends, who have given me enough courage to face all the odds and difficulties. Last but not the least, I would like to give special thanks to a friends who accompanied the tough days of writing thesis and preparing defense. Without the help of these great people, I would not have enough courage to finish this work.

Finally, I want give the largest gratitude to my family who is always supporting and pulling me out of plight. I want to give this honor to my two grandfathers who passed away in my graduate career, their wisdom is my most valuable treasure in my life. Also I want to thanks my little brother who is always standing on my side and giving me some great opinions. At last I want to thank my parents, your love is the greatest encouragement for me. The patience and the advices from you have been the strongest pillar of my life. I am totally indebted to your selfless love. Still there are too many people not mentioned but worth my honor. Thanks for the great friendship from these guys and I will cherish all these relationships in my whole life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

AC	Alternating Current
AFE	Analog Front End
ASIC	Application-specific Integrated Circuit
ATE	Automatic-testing Equipment
BGA	Ball Grid Array
BIST	Built-In Self-Test
BOST	Built-off Self-Test
CLB	Configurable Logic Block
DAC	Digital to Analog Converter
DC	Direct Current
DDJ	Data Dependent Jitter
DDR	Double Data Rate
DFE	Decision-feedback Equalizer
DFT	Design for Testability
DUT	Device under Test
FIFO	First In First Out
FPGA	Field-programmable Gate Array
Gbps	Giga-bit per second
HDL	Hardware Description Language
HSTL	Hi-speed Transceiver Logic
I/O	Input/output
IC	Integrated Circuit
JTAG	Joint Test Action Group

LFSR	linear Feedback Shift Register
LPM	Low-power Mode
LUT	Look-up Table
LVCMOS	Low-Voltage Complementary Metal Oxide Semiconductor
LVDS	Low-Voltage Differential Signal
LVTTL	Low-Voltage Transistor-Transistor Logic
MHz	Mega Hertz
MISO	Master Input, Slave Output
MOSI	Master Output, Slave Input
MUX	Multiplexor
PCB	Print Circuit Board
PCS	Physical Coding Sublayer
PECL	Positive Emitter-Coupled Logic
PCIE	Peripheral Component Interconnect Express
PISO	Parallel-In Serial-Out
PLD	Programmable Logic Device
PLL	Phase-Locke loop
PMA	Physical Medium Attachment
PRBS	Pseudo-random Binary Sequence
PVT	Process, Voltage, Temperature
RAM	Random Access Memory
RX	Receiver
SATA	Serial Advanced Technology Attachment
SCLK	Serial Clock
SERDES	Serializer/Deserializer

SIPO	Serial-In Parallel-Out
SMA	Sub Miniature version A
SoC	System on Chip
SPI	Serial Port Interface
SS	Slave Select
SSTL	Sub Series Transceiver Logic
TCK	Test Clock
TDI	Test Data In
TDO	Test Data Out
TMS	Test Mode Select
TX	Transmitter
USB	Universal Serial Bus
VCO	Voltage-Control Oscillator

SUMMARY

This thesis presents methods for developing a multi-function, high-speed (multi-GHz), and low-cost FPGA-based testing platform to replace the conventional automated test equipment (ATE) which have been widely used for digital test in past three decades. The development of semiconductor technology has outpaced advances in ATE testing capabilities, and the cost of ATE becomes the major expense of the semiconductor industry. These two major issues have caused significant problems for testing high-speed digital devices. Therefore the motivation of this research is to develop a more efficient solution and solve these issues.

The approach in this thesis is to utilize FPGAs as the general-purpose testing core and develop several specific-purpose testing modules on the basis of FPGAs. The FPGA-based arbitrary timing generator (ATG) module used the moderate-speed I/Os (typically 400Mbps) from either ATE or other testing platforms to serialize a high-speed data signal (3.2Gbps). This ATG board used the concepts of FPGA logics implementation, skew management, and multiplexing to synthesize continuous timing on-the-fly signal. The outstanding performance from this board has proven the primary idea of implementing the next-generation testing platform by using FPGAs. However, this ATG board still requires the control signals from ATE which does not qualify the goal of replacing ATEs.

In later sections we developed an independent FPGA-based testing platform without the host of ATE. This platform provides a wide-band (DC to 10Gbps) testability, multiple testing channels and compatibility of most current and future I/O standards. With the re-configurability of FPGAs, FPGAs are able to be re-programmed to handle different testing scenarios. The high-performance transceivers make FPGAs to have an even or better performance compared to current ATE systems. On this platform, four plug-in slots are reserved to provide power, normal I/Os and high-speed testing channels to custom-

designed extension boards. The on-board auxiliary communication interfaces and the control unit implemented inside the FPGA allow this platform be directly controlled by end-users and also to host plug-in boards. The power consumption of this platform including FPGA device and auxiliary chips is very low, so that it only requires typical power supplies and doesn't need a sophisticated liquid-based cooling system which save the total cost of equipment installment. These features have demonstrated the superiority of this platform over the existing ATE systems.

In addition to the performance and energy efficiency, this FPGA-based testing platform also allows plug-in boards to extend the testing functions which are not provided by the FPGA. We have showed the pin electronic (PE) testing and bi-directional data transmission (TX and RX) on a PE extension board. The board not only includes flexibility in transmission but also provides low amplitude data detection and recovery. Another Ultra-High-Speed plug-in module was developed to speed up the output data rate to 40Gbps, which allows this FPGA-based testing platform to keep the same pace with the development of semiconductor industry. The relative easy-design extension boards have increased extra competitive strength of this platform over other testing systems.

The advantages of the methods addressed in this thesis are significant. The high-speed, low-cost, and energy saving FPGA-based testing platform combined with the flexible extension modules will become the mainstream of digital testing and eventually replace the existing ATE systems in the future. Therefore the contributions made in this thesis have the potential to entirely change the way we have used to test digital devices in the semiconductor industry.

CHAPTER 1

INTRODUCTION

The objective of this research is to develop a high-performance and economical platform to test high-speed digital devices at multi-GHz speed rates. Due to the development of semiconductor process and the well-known prediction of Moore's Law [1], the complexity and performance of digital devices have had an exponential growth for the recent 30 years. Therefore, testing high-performance digital systems is becoming a challenging issue. Many solutions are provided to solve this problem. However, these solutions are expensive, inefficient, and consumes a lot of power. In this research, we present an efficient, low-cost and low-power approach to test high-speed digital systems.

1.1 Motivation

For the past three decades, ATEs have been generally applied to high-volume digital testing. In the meantime, the performance of ATE has improved a lot and new capabilities and features have been added to extend its applications. However, ATEs' performance still cannot keep up the speed of semiconductor development [2]. While the commercial digital systems are running at 16Gbps (PCIE 4.0) [3] and 25 or 50Gbps digital systems will soon appear [4] [5] in the market, most ATEs still remain at a relative low-speed data rate (typically below 3.2Gbps) and become out-of-date for testing the most cutting-edge designs. Furthermore, the high cost and the high power consumption are the other concerning issues for traditional ATE systems. Although ATE is still a popular test platform for digital devices today, the high price and power issue has increased the total cost of high-speed testing. Most of ATEs also lack the flexibility of testing different digital systems. If digital designers want to test other digital devices, very limited modules can be selected on existing ATE systems to extend the testing functions. Therefore, developing a

high-speed, adaptive and low-cost testing platform is urgent to replace these old ATEs and provide a more efficient testing method for future digital devices.

1.2 Approaches

The main approach we adopted in this thesis is to use FPGAs as the testing core and solves the issues we have mentioned in motivation section. FPGAs are adaptive, low-cost, and configurable digital devices. Most digital-system designers use FPGAs to design reliable embedded systems, implement digital logic, and verify new-developed algorithms. Because of the development of semiconductor technology, the high-end FPGAs can operate at ~800MHz [6], which enables the potential for FPGAs to be applied to the area of high-speed digital testing. With the built-in high-performance serializer/deserializer (SERDES) and transceiver components, FPGAs are able to transmit and receive multi-Gbps signals [7]. Therefore, the performance of modern FPGAs can be used to test commercial digital device or high-speed designs in the coming future. Furthermore, the re-configurability makes FPFAs very flexible to fit in different testing scenarios.

In this thesis, a FPGA-based arbitrary timing generator is first introduced to prove the idea of using FPGA on digital testing. This generator provides 3.2Gbps timing-on-the fly testability as an extension module of ATEs, which not only extends the product life of those out-of-date ATEs, but also provides new function that haven't been developed on the most advanced ATEs . Later a high-performance FPGA-based test platform is presented for high-speed digital testing in order to replace the host ATE. The most cutting-edge 28nm-process FPGA from Xilinx Kintex-7 family is implemented as the main controller and the test pattern generator, fabricating with impedance-controlled design PCB boards, auxiliary communication ICs and high-bandwidth I/O connectors on an FPGA testing board. Furthermore, specific-purpose testing modules can be designed and plug-in on the main board, which extends this platform to different testing environments without re-designing the entire system. Also, this platform requires very low power as compared to

ATE so that expensive cooling systems and power supplies are not required. Therefore the FPGA-based testing platform is possible to replace the features of ATE and offering more flexibility and adaptability with economical cost.

1.3 Thesis Organization

This thesis is organized as follows: In Chapter 2, a brief history of recent testing methods including ATE systems and BIST is introduced, the pros and cons of these two methods are also well-discussed in this section. The following part of Chapter 2 introduces the development of modern FPGAs, and the features in FPGAs that are possible to be utilized in digital testing. In Chapter 3, a FPGA-based extended ATG module for ATE is presented to show the potential for using FPGAs to synthesize complex timing waveform. Chapter 4 and Chapter 5 introduce an independent FPGA-based testing platform and its high-performance extended modules. The experimental results of these two chapters are convincing that our method is able to reach the performance and solve the issues we have. In Chapter 6, we first briefly summarize the achievement of this thesis, and followed by the limitations and future works of current designs, in the last part the contributions and conclusion are addressed to conclude this thesis.

CHAPTER 2

BACKGROUND AND HISTORY

Hardware testing and evaluation continues to be a challenge for high-performance digital devices. For the past 40 years, ATEs with multi-channel I/Os have provided a stable method for digital testing. However, as the data rate of modern digital systems goes up to several Giga-bits per second (Gbps), the approach of generating high-speed testing patterns becomes more complex and harder to be implemented. Although ATE in the meantime is also progressing to achieve higher speeds, the exponentially high-upgrading cost and the testability still present difficulties for testing the modern digital systems. Therefore, it is needed to design a new testing architecture to replace the conventional ATE testers, and apply this design to those high-performance digital systems in the current and future market. In order to solve this issue, build-in self-test (BIST) is a common solution in current silicon industry. However, this approach lacks the flexibility and increases the time of developing new products. FPGAs are adaptive device and very flexible in re-configuration, which are considered to be a very-potential candidate to be used for implementing a new testing platform. In this chapter, a brief review of ATE history, BIST approach and its capabilities along with FPGA development and characteristics are described.

2.1 Automatic Test Equipment

Traditional ATE is a common and reliable approach to test different digital systems. Most testing engineers have used ATE to do all kinds of testing purposes for the last few decades. Nicholas DeWolf, the chief engineer at Semiconductor Manufacturer Transitron, teamed with Alex d'Arbeloff in 1961 to found Teradyne in Boston, they built the D133 diode tester, one of the first commercial test systems to use semiconductors in place of vacuum tubes. In 1966, DeWolff and Milt Collins designed the model J259 based on the

Digital Equipment Corporation PDP-8 mini-computer became the first computer-controlled test system in the industry [8].

The first-generation ATE was very primitive compare to its modern successors. Later in 1970s it started being developed to achieve more complex functions, ATE systems with those modern testing capabilities were introduced to test radio frequency (RF), mix-signal, optical and digital components. In the late 80s, the speed of digital components was explosively increased due to advanced semiconductor technology [1]. Therefore the manufacturers of ATE were seeking different approaches to increase the speed of their test systems. Multiplexing was one of the approaches being introduced to achieve higher speed at that time. Tektronix S-3295 and Megatest MegaOne used this concept to double the output frequencies of their products [9]. However, these achievements still did not allow ATEs keep the same pace with semiconductor development. These ATEs only operated at low hundreds MHz speed but the very high-end digital devices were running much faster than this speed. Furthermore, the timing resolution of ATE was only in the nanosecond scale and this accuracy was way rough in many testing settings.

In the 1980s, a revolutionary approach was introduced into the industry and change the existing architecture of ATE systems. Until now, most of ATE systems used a "shared resource" architecture, which was very competitive and inexpensive when only a few channels were implemented. However, if more testing channels are required by users, a lot of switching and multiplexing circuits would be demanded to fulfill the sharing purpose. The rising hardware requirement not only increased the complexity and the cost of ATE system, but also made the testing calibration and programming procedure become dramatically difficult. To solve the above issues, ATE manufacturers introduced "per-pin" architecture to replace outdated "shared resource" architecture. Under this architecture, each testing pin or channel included its own dedicated pin electronics resources such as pattern generation, timing (phase) control and voltage-level or voltage swing. This feature eliminated those complex switching circuits and phase-variation issues between channels,

and it was more efficient and easier to be used since each channel could be controlled individually. The most brilliant part of this architecture was that users could initially keep a relative low-cost base system and expand additional channels or functions as they are required in future testing.

Hewlett Packard released the first "per-pin" architecture series HP83000-F660 in early 90s. Despite the fact of the performance of those testers could not catch the speed of devices, still a lot of new features were added or improved compared to old models. The new designs included much larger amounts of memory, sophisticated test pattern algorithms, elaborated parametric measurement units and relative low cost when adding additional modules. However, those testers still consumed a lot of power and generated too much heat. A very efficient liquid-cooling module was required and therefore raised the total cost of installment and operating these ATE systems. Since a lot of new features were added to ATEs, the price of ATE didn't benefit from semiconductor development and drop significantly. ATE per-pin cost for high-end system was still around $3000 -$10000, and the typical 256-channel system cost from $1M to $5M.

The manufacturers started to introduce a new "open-architecture" test platform in early 2000s [10]. In this architecture, testers could be fully customized from motherboard, CPU, memory and other auxiliaries to build an application-specific system. Therefore customers could purchase a basic system containing the function cards they needed initially. Once a high-end testing was demanded, this system was allowed to add higher performance or different testing purpose cards to extend testing ability [11]. Verigy (former HP's testing division) developed this kind of base system called V93000 SOC (the replacement of outdated HP83000). Base on the basic system, customers could add various function cards such as digital nano-electronics testing, high-speed digital testing and mix-signal testing [12]. Although the open-architecture had saved a lot of cost for ATE systems, the demand of higher performance, better accuracy, increasing memory requirement and other advanced testing features had offset the cost it saved.

The latest development of ATE industry was protocol aware (PA) architecture [13]. The concept for PA-ATE arose as higher levels of integration allowed the manufacture of diverse devices using technologies such as system on chip (SoC). Device manufacturers with substantial intellectual property (IP) libraries could develop a system with diverse IP blocks in a single process on a single die. In this method, each block could have different signaling protocols and each protocol would require its own testing strategy, thus testing the whole device on global level would be very inefficient. To solve this problem, PA-ATE essentially emulated chip I/O at the protocol level. Programmable interfaces in PA-ATE were used to perform real-time state detection to handshake with devices using its native protocol. Test strategies were developed for each protocol, and once a communication link to the device is established, testing would perform in a more efficient manner [14] [15].

The multi-channel I/O capability, multi-function extended boards and relative high-speed output data rate make ATE still a powerful platform on digital testing today. However the high cost and large power consumption makes it difficult to apply for all the digital testing environments. Furthermore, although the speed of ATEs can go up to multi-Gbps, digital devices have benefited more from modern semiconductor process and are able to exceed this speed easily. Therefore a more efficient approach to digital testing is needed to solve above issues from current ATEs.

2.2 Built-in Self-test (BIST)

Test stimuli and response evaluation has traditionally been performed by an external ATE system. As the digital devices have become more complex and access to embedded components become more difficult, the current external ATE could not perform comprehensive testing. To solve this problem, several approaches of design for testability (DFT) were developed to compensate the limitations of ATE on those devices. The first DFT method included building additional module into devices to enable external testing, the most common approach of this style DFT is scan-based design which allows the access

and control of device components connected to internal registers through scan chains. Another popular DFT method of modular testing was later developed to test similar module in explosive-growing complex devices, this method had reduced the testing generation time and improve the efficiency of the original DFT method. Among those DFT methods, the most popular way is to design internal test modules onto the device itself as the technique called of built-in self-test (BIST). This technique was first applied to test memory devices in late 1980s [33]. Additional on-chip circuitry added on to the circuit-under-test that generates test stimuli and evaluates responses to verify the function of the circuit.

BIST method is generally specified into two major types: online and offline. Online BIST is designed to function when a device is in normal operation mode, in other words, it is able to detect errors in real time. Offline BIST is designed to perform testing functions when the device is not in normal operation mode, such as on power up procedure [34]. Both types of BIST require very similar architecture compare to the under-testing device. The main components of the BIST are test pattern generators (TPGs) and output response analyzers (ORAs). A BIST controller component controls the operation of TPG and access outputs from ORA to determine the test results. These components are usually built by using registers and finite state machines (FSM).

The complexity of BIST depends on the amount of test patterns required and the number of circuit-under-test. BIST has been an efficient method of DFT for past 20 years [35] [59] [60]. Miniaturization of transistor features has allowed additional circuitry to be built into devices without much imposition on the devices performance. However, as device complexity increased exponentially, employing regular BIST method has become a challenging issue. Therefore in 1999, Credence Systems Corp. introduced the concept of built-off self-test (BOST). In this DFT technique, additional self-test circuitry is added to a device, but built off the device such as on the same load board. Typically a field-programmable gate array (FPGA) is used to test the control circuitry and generate test patterns. The BOST technique provides tremendous flexibility, as it removes physical

limitation of BIST. Furthermore, the use of those cutting-edge FPGAs can develop elaborate and high-performance testing solution.

Although BOST seems to remove the most drawbacks of BIST approach on complex circuits, the other existing issue is it still required full-customized circuit design for a certain DUT. Therefore it makes this technique wastes a lot additional time through traditional IC design and manufacturing procedure, which increases the invisible testing cost of the new devices and delays the schedule of new devices debut on the market. In short, BIST/BOST is still not a very ideal testing method to be applied millions of under test devices in this fast-growing industry, a more efficient and save-time method needs to be implemented to IC design industry.

2.3 Field-programmable Gate Array Development

A field-programmable gate array (FPGA) is an integrated circuit designed to be configured by a customer or a designer after manufacturing. The configuration of FPGAs is generally specified by using a hardware description language (HDL) and it gives FPGA flexibility to be programmed to various digital applications. Therefore utilizing FPGA becomes a popular approach for digital designers to verify the primary concept and implement circuit designs in a short time with low expense. The FPGAs' appearance has boosted the speed of announcing new digital products in IC design industry.

For fabless semiconductor industry, the mainstream products started from transistor-level circuits to integrated circuits, then followed to application-specific integrated circuit (ASIC), and finally came to programmable logic devices (PLD). PLDs started developing from the leading semiconductor industries such as Motorola, Texas Instruments, and IBM in the early 1970's. But it wasn't until Xilinx brought us the first FPGA in the late 1980's that PLDs crossed paths with the ASIC world [14] [15].

The appearance of FPGA technology not only cut millions dollars cost for semiconductor manufacturing, but also dramatically reduced the time of delivering new

products to market. This was a revolutionary change in digital design, and therefore most of the digital designers could implement and verify their ideas easily. Xilinx was founded in 1984 [16], Ross Freeman and Bernard Vonderschmitt invented the first commercially viable FPGA in 1985 – the XC2064. Seiko started manufacturing XC2064 for Xilinx in 1985 using a mature 1.2m process. The XC2064 had 64 configurable logic blocks (CLBs), which with two 3-input lookup tables (LUTs) built in each and the operation frequency of 18MHz. XC2064 and its successors had lead Xilinx continued to be unchallengeable and become the market leader of FPGA from 1985 to the mid-1990s.

The 1990s was an explosive period of time for FPGAs, both in sophistication and the volume of production. After a decade of exploring possible markets, FPGAs found their way into commercial embedded systems, automotive, and industrial applications. In the decade of 2000s, the capacity of FPGA continued to grow and the functions of FPGA were enhanced to be applied to more areas. Both leading companies: Xilinx and Altera have announced the FPGA with more than 300,000 CLB with 800MHz operation speed in late the 2000s [17] [18] [19]. Also more memory, digital signal processors and high-speed I/Os are built-in to provide FPGA more flexibility in various applications.

A recent trend for FPGA in early 2010s has been combining the logic blocks and interconnects of traditional FPGAs with embedded microprocessors and peripheral components to form a complete "system-on-a-programmable-chip" [20]. Examples of such hybrid technologies can be found in the Xilinx Zynq™-7000 all-programmable SoC, which includes a 1.0 GHz dual-core ARM Cortex-A9 MPCore processor embedded within the FPGA's logic fabric [21]. Other examples under similar architecture is Altera Arria V FPGA, Atmel FPSLIC [22] and Actel SmartFusion [23] have been delivering into the market and applying in different areas. The high level of integration on this architecture helps to reduce power consumption and dissipation, this change leads to a lower parts cost, a smaller system, and higher reliability since most failures in modern electronics occur in the connections between chips on PCBs instead of within the chips themselves.

High-speed I/O interface [6] [7] in FPGA is another spotlight on recent FPGA development. By the concept of serializer/deserializer (SERDES) [24] [25], FPGA with high-performance transceivers is able to transmit 10Gbps or higher speed data signal. SERDES is the component that facilitates the transmission of parallel data between two points over serial streams, reducing the number of data paths and thus the number of connecting pins or wires required is cut. Therefore SERDES enables the movement of a large amount of data point-to-point while reducing the complexity, cost, power, and board space usage associated with having to implement wide parallel data buses. Most SERDES devices are capable of full-duplex operation, meaning that data conversion can take place in both directions simultaneously. SERDES are widely used in the market of Gigabit Ethernet systems, wireless network routers, fiber optic communications systems, commercial I/O standards [25] and storage applications [24].

Many commercial high-speed I/O standards such as USB3.0, DDR-3, PCIE and SATA are running up to 10Gbps and different I/O standards usually comes with different encoding/decoding algorithms in data transition. The traditional testing cost is very high at Gbps data rate, and different encoding/decoding methods also increase the complexity of testing these standards on an individual ATE system. With the features of high-performance SERDES and multi-channel ability, also low cost of manufacturing compared to current ATE systems, FPGA transceivers can be designed in a high-speed test system and be applied to high-end digital testing. Also, the re-configurability of FPGA makes it very flexible to be programmed for different testing purposes [61] [62] [63] [67]. Therefore using economical FPGA and its transceivers has become a popular approach to replace the traditional testing platforms. Some of testing modules [62] [63] of ATE and other testing platforms are design by FPGAs to benefit the advantages from FPGAs, but the critical testing core is still dominated by ATEs. There should be a wider and deeper utilizing of FPGA in the high-speed testing that has not been well-developed yet.

CHAPTER 3

MULTI-GIGAHERTZ ARBITRARY TIMING GENERATOR

3.1 Introduction

For past two decades, digital-circuit designers have relied on software-based simulation to verify logics and optimize the performance of their designs. Most simulation software provides unlimited flexibility in defining signal waveforms, in other words, designers can define any speed, edge and phase of the signals for their own testing purpose. However, when these tests are applied to the real world, for instance, device-under-test (DUT) hardware testing by using ATEs or other pattern/signal generators, the hardware imposes a lot of limitations on actual signals [57].

Most of hardware-based digital testing today is dominated by ATE systems [26]. These machines provide high-speed (up to several Gbps), multi-channel ability and very limited edge configurability (phase adjustment). Modern ATE systems are implemented by the concept of multiplexing and double-data-rate (DDR) to achieve higher data rates than the master clock rate. However, the limitation is that the resulting high-speed serial pattern sequences are generally produced at a fixed frequency within a continuous test sequence and are hard to be changed to other frequencies [27]. This constraint still cannot be avoid at or near the maximum speeds of the most ATE nowadays.

At moderate speeds (lower than 1Gbps), ATE may be able to synthesize limited "timing-on-the-fly" behavior [28] [52]. The period of the test signal is usually an integer multiple of the ATE master clock. Therefore true timing-on-the-fly capabilities are very limited, with many timing constraints above the moderate (lower than 1Gbps) frequency limits. ATEs are also limited by using simple look-up tables of timing values to implement the desired delays [50] [51]. The re-loading of timing values is often accompanied by extra "dead cycles" wherein the test signal logic states are temporarily held fixed for some

number of ATE master-clock cycles [53] [54]. Therefore, a multi-gigaherz Arbitrary Timing Generator (ATG) is presented in following chapter, which is able to synthesize timing-on-the-fly at 3.2Gbps with 10ps phase-shift resolution.

3.2 ATG System Overview

The ATG is designed using a Xilinx Spartan-6 FPGA as a central control unit combined with high-speed logic (HSL) that produces the real-time delays and handles the formatting serialization of multi-GHz signals, and the interface to host ATE. The top-level architecture of the entire system is shown in Figure 3.1.

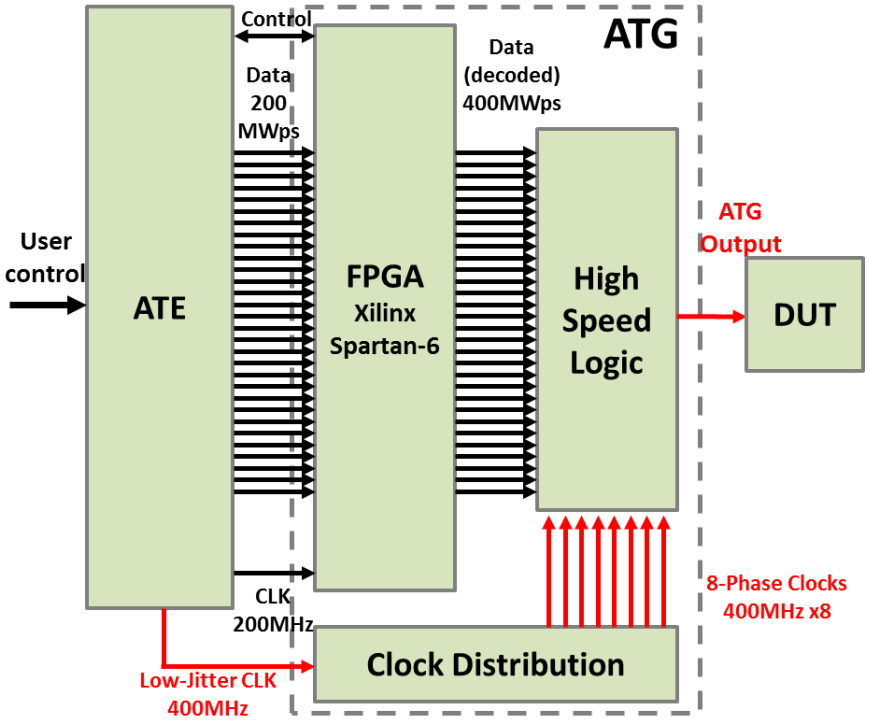


Figure 3.1: ATG block diagram.

The algorithm that generates the control signals for HSL was implemented inside the FPGA. With the deep-pipelined and high-volume parallel calculations inside the algorithm, the FPGA is able to control the HSL to synthesize Gbps signals. The clock distribution, which is built by high-bandwidth fanout and delay chips, will take a reference clock from the ATE host and distribute the reference clock evenly into 8-phase clocks for

high-speed logic. The “high-speed” (HS) section is composed of high performance and low-jitter components which are used to generate the arbitrary waveform. The FPGA takes the pointers from ATE (200 Mega Words/s) and operate a complex parallel calculations (200MHz) then output required control signals (double date rate to 400Mbps) for the logics (flip-flops, delay logics and XORs) in HS section. In the final stage, the HS section takes those signals and 8-phase clock (400MHz) to synthesize the final arbitrary waveform. The target spec and capability of the ATG is shown in Table 3.1:

Table 3.1: Specification of ATG

Data rate	DC to 3.2Gbps (normal mode, w/ full function)
	6.4Gbps (burst mode, w/ limited function)
	10Gbps (burst, mux mode, w/ limited function)
Min. pulse width	$\leq$ 100ps
Timing accuracy	~10ps
Jitter	~30ps peak to peak
Voltage swing	450mV single-ended (peak to peak)
Data pattern	Pseudo random binary sequence (PRBS), clocked pattern, user-defined
PRBS support	7, 15, 23, 31

3.3 ATG Hardware Design

3.3.1 High-speed Logics and Clock Distribution

The key components of HSL are edge generators, high performance flip-flops and serializing XOR logics. Edge generator is basically form by a skew-management component, with programmable control ports, the components is able to perform phase shift at output side. The component used in this design is Micrel SY89296U programmable

delay-line chip [36]. This component is able to perform 10ps to 9.2ns delay with 10ps resolution, and the minimum propagation delay is about 3.2ns, the schematic of this delay component is shown in Figure 3.2. The /EN control is used to enable/disable the input of the chip. Latch enable (LEN) is utilized to sample the programmable 10-bits wide digital delay bus D [9:0]. To either pass a specific gate delay (GD) value or bypass that GD is determined by each delay bit. With programing D [9:0] and the latch enable (LEN) signal, the precise delayed signal associates with the delay-control bits will occur at output.

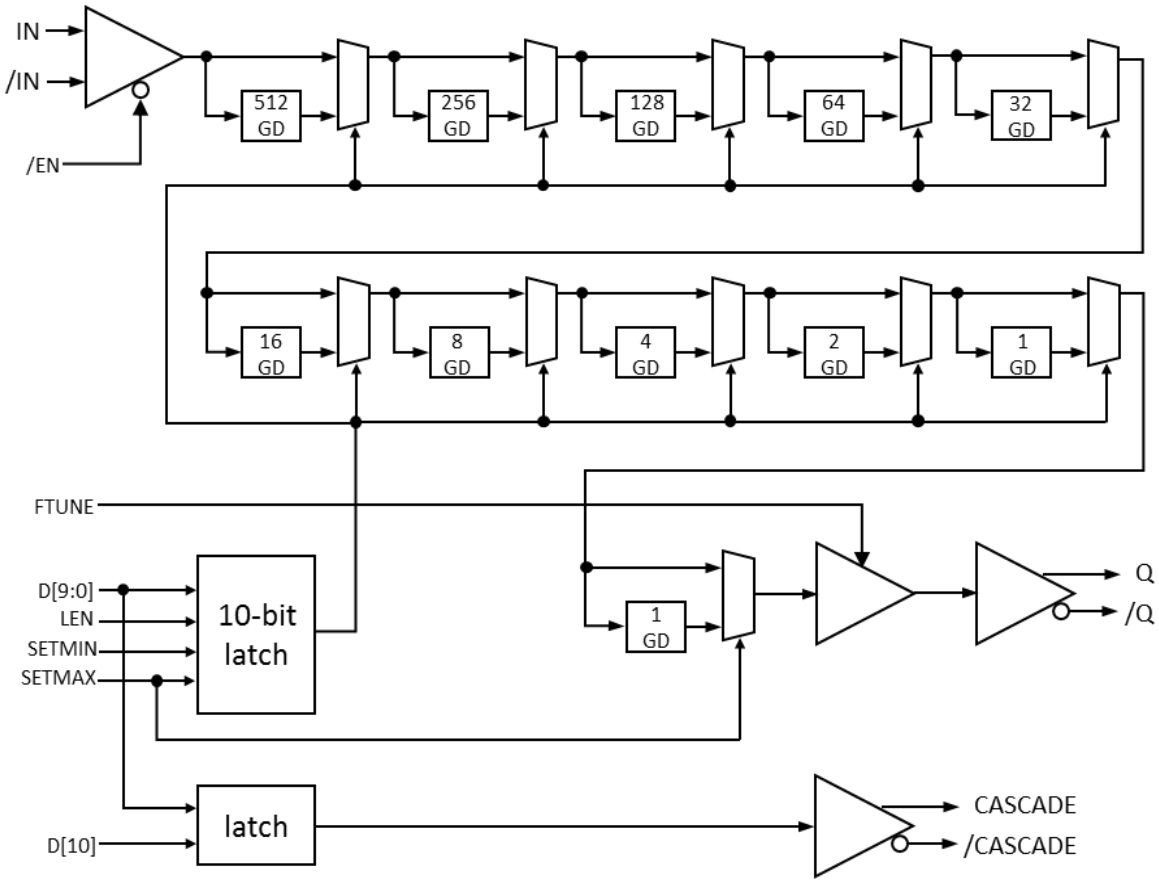


Figure 3.2: Block diagram of the delay chip

The timing chart of this delay chip is shown in Figure 3.3. LEN signal initially stays at high level to avoid programming. After the target programming edge arrives and all 8 delay control signals remain stable, the input LEN will be de-asserted to allow the delay value (D [9:0]) programmed to the chip. The LEN requires to be held for at least 0.5ns for programing process, then it is asserted back to high level after the programing is completed,

and keeps staying high until next programming edge and delay controls arrive. The output edge is delayed by propagation delay value (~3.2ns) plus target delayed value compared to input edge. The whole process requires precise cycle and phase control on the delay signals and LENs, all these control signals are calculated and generated in FPGA.

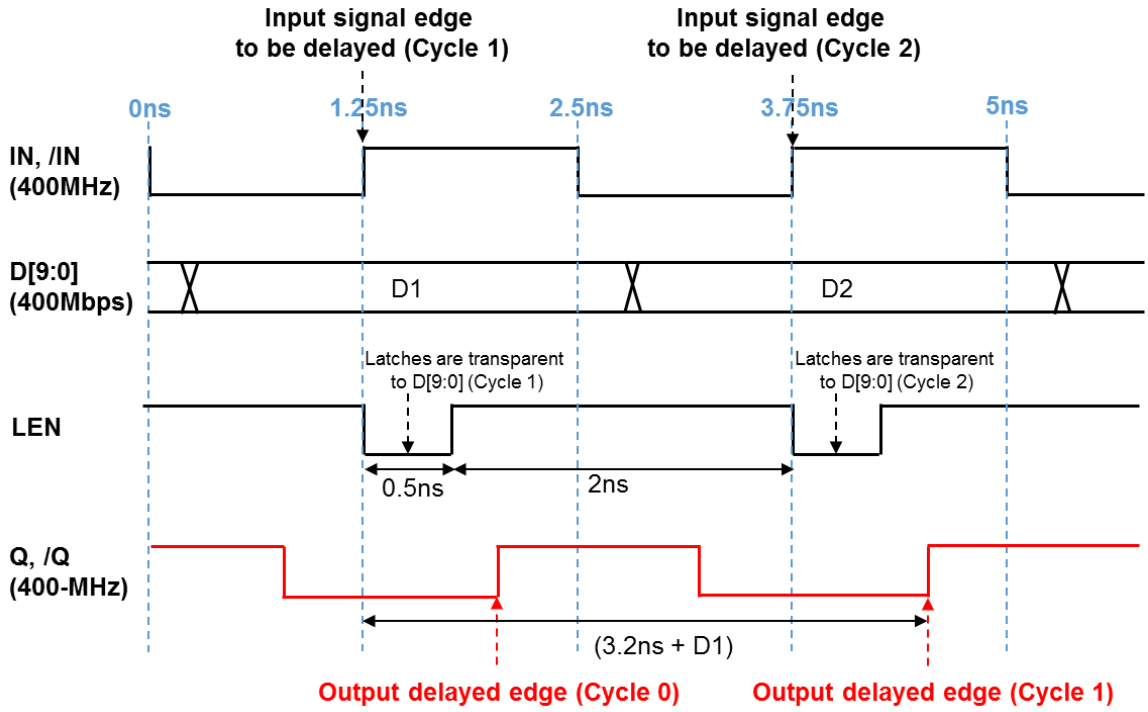


Figure 3.3: Timing chart of the delay chip

The fine tuning control port in this delay chip that allows users to adjust the propagation delay is very critical to improve the accuracy and synchronize different delay components [55] [56]. Since there exists process variation among the chips, and also the voltage and temperature issue (PVT), the propagation delay is not identical for different chips. The propagation delay play the role as reference "0" delay for the waveform. In fact the delay chip delays the input about 3.2ns even the delay value is programmed to all 0s. However if every delay chip delays the same amount, then we can consider the whole waveform delays 3.2ns and see this delay as relative "0". Therefore if the propagation delay is not identical among those chips, then delay chips are not able to generate edges base on the same reference. The generated edges will be serialized to format final waveform, if the

reference “0” is not identified, then the edges of waveform might be off even correct delay values are programmed. The adjustment range of this propagation delay is 50ps, the propagation delay vs. control voltage is shown in Figure 3.4.

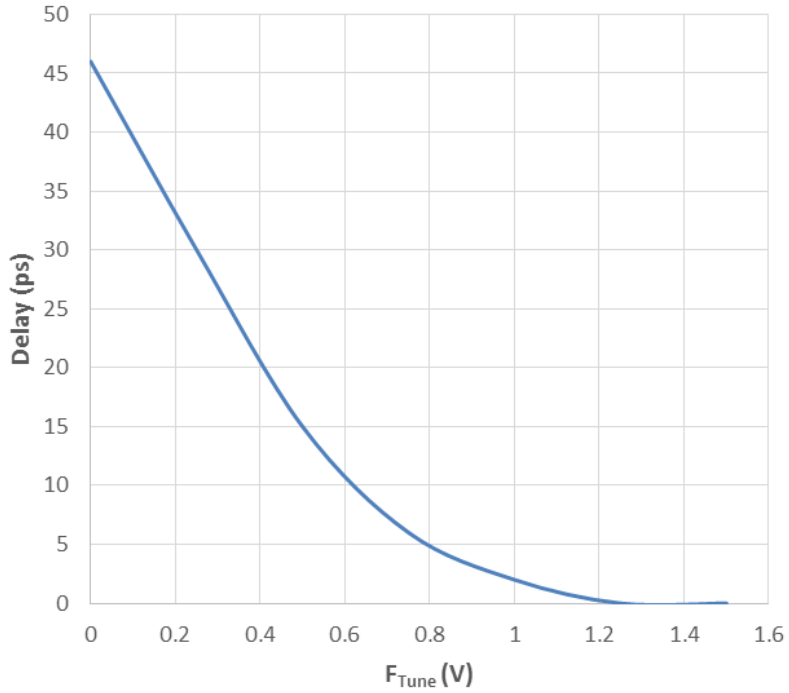


Figure 3.4: Adjustment range of fine-tune delay control of the delay chip

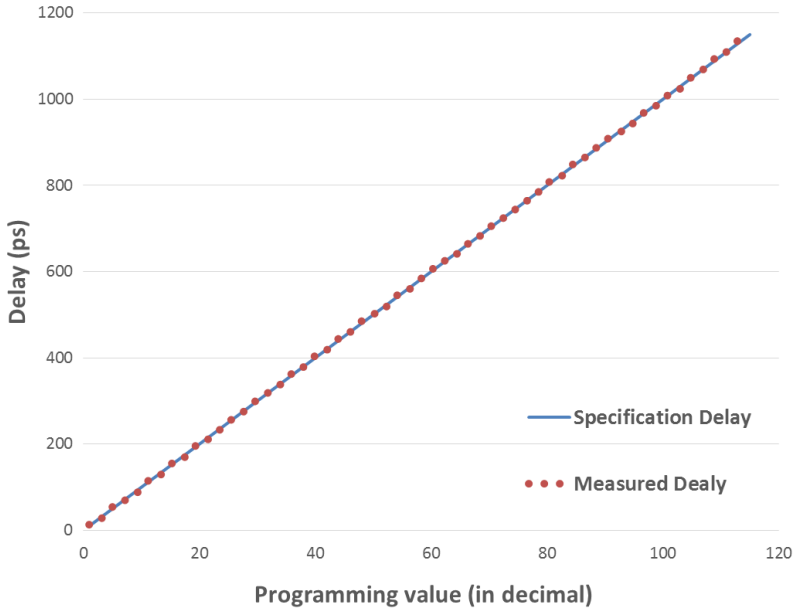


Figure 3.5: Delay-linearity verification of the delay chip

The delay-value linearity verification of the delay chip is shown in Figure 3.5. We compare the actual delay time with the delay time claimed in the datasheet of SY89296U. The measurement start from 10ps delay (control code = '0000000001') and end at 1150ps (control code = '0001111111') which covers the delay range we will use in ATG system. The result of this experiment shows an excellent delay linearity in this device.

The high-performance flip-flop (FF) of On Semi NBG53 [37] is used to synthesize the pattern data with reference clock. The basic concept is to use the rising edge of delayed clock from delay chip to sample the input data to generate a delayed version of data signal. The timing chart is shown in Figure 3.6. Notice that the setup time needs to be larger than 30ps to ensure the sampling process functions correctly, and the propagation delay (data input to output) is about 210ps. This FF chip supports up to 10GHz sampling clock which ensures enough bandwidth in our application (400MHz).

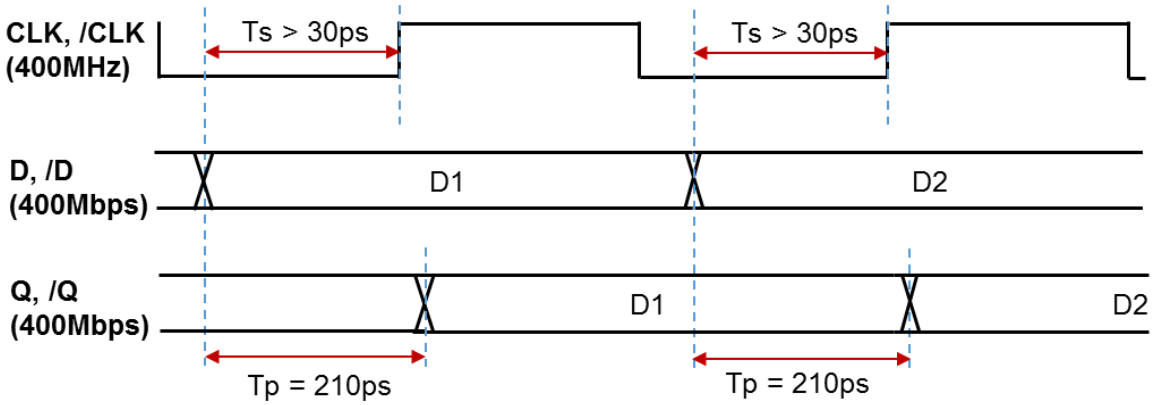


Figure 3.6: Timing chart of high-speed flip-flop

The last critical component of HSL is XOR logics. Exclusive-OR is a popular method to implement serialization. In ATG system, eight moderate-speed (typically 400Mbps) channels will be serialized to a high-speed (3.2Gbps) channel. In order to implement this precise work, a SiGe XOR (HMC745) [38] component from Analog Device is selected. HMC745 is able to carry up to 13Gbps signal with sharp rise/fall time (~20ps) and 0.2ps RMS random jitter (RJ). Since this XOR is the final stage of the high-speed logics, the rise/fall time of this chip affects the full transition of the final waveform. Also

three stages of serialization is required for serializing total eight channels, the random jitter is accumulated as the signal passing through stages. Therefore a sharp rise/fall time and low RJ device is required to ensure signal quality for Gbps data transition.

Clock distributor not only fanout the reference clock to 8 identical copies for 8 edge generators, but also spans 8-channel clocks to evenly-spaced phases of 2.5ns cycle. In the very beginning, a clock fanout structure is used to deliver the reference clock to 8 identical clocks. The fanout chip NB7L14M with 10GHz bandwidth and <0.5ps RMS jitter is used to minimized the jitter on those 8 clocks. Since this low-jitter component is only capable to fanout 4 channels, a two-stage structure of 3 total chips is required to achieve 8-channel fanout. 8 clocks then are delivered to phase skew module which is also built by SY89296U delay chips to generate 8-phase clocks. Each delay component will keep a fixed delay value and each of delay chip delays 312.5ps (2.5ns of 400MHz period divided by 8) more than previous one. The timing relationship of the reference clock and 8 delayed clocks form clock distributor is shown in Figures 3.7.

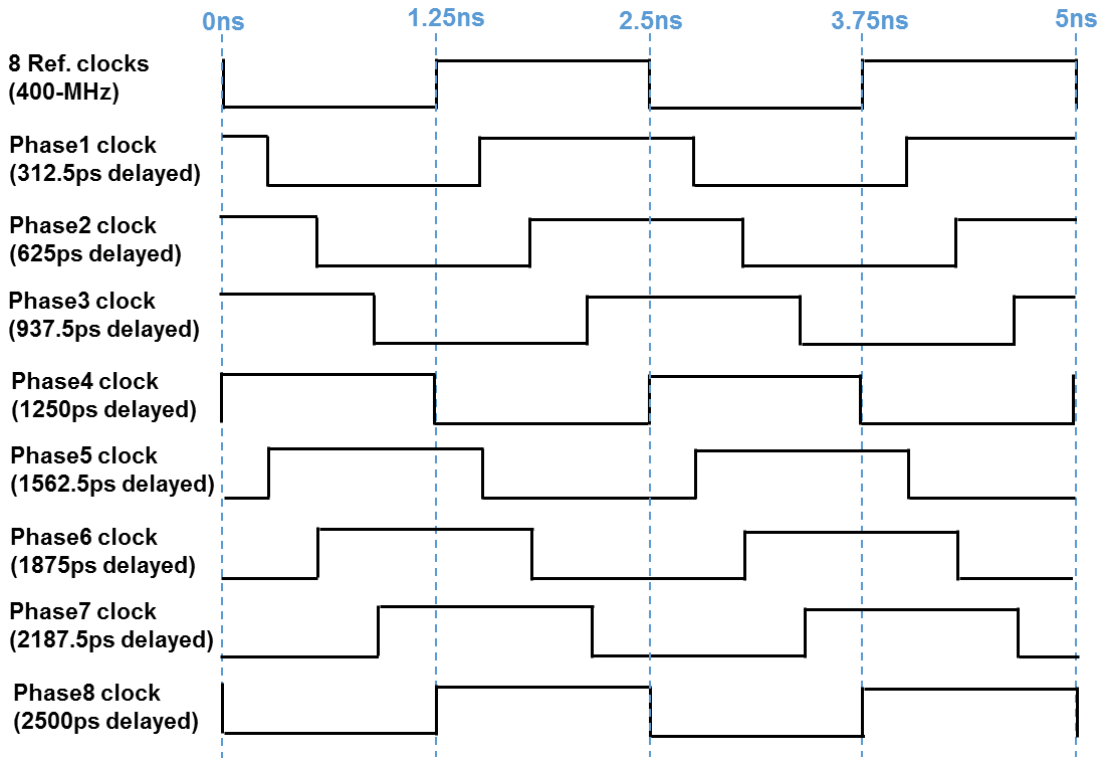


Figure 3.7: Timing relationship of reference clock and 8-phase clocks

3.3.2 FPGA evaluation

The FPGA core plays the role of interface of ATE and extension board, and generating the precise controls signals for HS logics to serialize final waveform. Therefore the selected FPGA must provide the capabilities to perform these two critical characteristics. The four common criteria used to evaluate the appropriate FPGA is:

1. I/O compatibility
2. FPGA performance
3. Logic capacity
4. I/O counts, physical size, price and power consumption

The IO compatibility between ATE, FPGA and HS logics is the most critical criteria in this ATG application. The ATG extension module is not designed for specific ATE or specific testing systems, therefore the FPGA should be able to provide the flexibility of supporting most available I/O standards in the market. Also the HS logics transmit and receive differential high-speed signals and accept single-ended control signals, the selected FPGA should also be compatible to these I/O standards as well.

The performance of FPGA dictates the maximum speed of this module. While all the high-speed components are able to running at multi-GHz, the FPGA throughput is actually the bottleneck of this ATG system. A higher-performance FPGA is always preferred, however higher performance usually comes with the issues of higher price and power consumption issue which we always want to avoid for a low-cost system.

The algorithm of this ATG requires a heavy-loading parallel calculations, also the data generation/control logics, registers and memories are all implemented inside the FPGA. While selecting a FPGA to accommodate all these necessary logics to perform desired function, the logic capacity is another critical issue of consideration.

The last criteria is mutually associate to previous three. With higher performance and higher capacity, the price and power consumption defiantly increases. A better I/O compatibility and larger capacity come with lager physical size and I/O numbers. As we

mention before, flexible I/O compatibility and higher performance are always having the highest priority. In the meantime we need enough capacity and I/O channels of implement the logics and furthermore we don't want large power consumption and high price. Thus a balance strategy need to be performed here to match all necessary requirements.

Spartan-6 family is a high cost-performance ratio FPGA series introduced by Xilinx, this series is built by 45nm process with three speed grade (-1, -2, and -3) [39]. The exact FPGA model we selected is XC6SLX45-3 which provide maximum speed (~600MHz, register to register) in Spartan-6 family. The first criteria of deciding this FPGA is I/O compatibility. This device supports most available differential and single-ended I/O standards in the industry, which is detailed in Table 3.2 [40]. The wide I/O support of FPGA is compatible with both ATE platform and high-speed components on the board. In this research, single-ended SSTL2.5 Class II standard is mainly used to receive ATE outputs and control high-speed devices.

Table 3.2: Spartan-6 Supported I/O standards

Single-Ended Standard	**Differential Standard**
LVTTL	LVDS (2.5 and 3.3V)
LVCMOS (3.3V, 2.5V, 1.8V, 1.5V and 1.2V)	BLVDS (2.5V) Mini LVDS (2.5V and 3.3V)
PCI (33 and 66 MHz)	RSDS (2.5V and 3.3V)
Serial bus (I2C, SMBus)	PPDS (2.5V and 3.3V)
SDIO	LVPECL (2.5V and 3.3V)
DDR	DIFF. DDR
HSTL 1.5V and 1.8V (Class I, II and III)	DIFF. HSTL 1.5V and 1.8V (Class I, II and III)
SSTL 1.5V, 1.8V, 2.5V and 3.3V (Class I and II)	DIFF. SSTL 1.5V, 1.8V, 2.5V and 3.3V (Class I and II)

The I/O performance of Spartan-6 qualifies ATG target (to serialize multiple hundred-Mbps signals to a Gbps signal) and also match the output frequency of typical ATE system (660MHz in HP83000-660i). This FPGA also provides up to 43K logic cells (each logic cell is form by a 6-input look up table) for digital design and up to 2Mb block memory for data storage. To estimate the logic cells needed in the ATG algorithm, the place and route implementation was simulated under Xilinx ISE Design Suite software [41]. The result of simulation shows 60% usage of total FPGA resource to implement ATG algorithm. Furthermore, this FPGA supports up to 316 moderate-speed I/Os [39]. Consider the ATE pointers (6*8 = 48 bits), delay chip control signals (10*8 = 80 bits), input data pattern (16 bits) and output data pattern (8 bits), total ~150 I/Os are used for ATE and HS connections. Therefore I/O resource is still plenty for this ATG. From all aspects, Spartan-6 matches all the spec criteria in the ATG. Also consider the economical price of $80 and the low power consumption of 5W, this FPGA should perform the best cost-performance ratio to implement this ATG board.

3.3.3 Skew management

The control signals of the components used in HS section require a high-resolution timing management. The timing charts of high-speed logics from Chapter 3.3.1 have proved that a bad skew management for those control signals might fail the operations of those components. An easy way to control the phase difference is to adjust the length of the transmission path on PCB to match the timing requirement. However, this is not a reliable method as it might be affected by the PCB manufacturing variation and material selection, and furthermore a minor adjustment might be needed so the controllability of the trace length is needed. In this ATG system, the skew management inside FPGA is introduced to solve this issue. Figure 3.8 shows the block diagram of built-in delay line of the FPGA programmable I/O components [40]. An 8-bit delay value allows delays from 0 to 255 taps to be achieved in this component. The three LSBs of this value control the

starting point of a ring oscillator. The oscillator is triggered by the arrival of an input signal, and its output clocks a 5-bit counter from 0 to 7 tap delays later. The 5-bit counter is preset by the five MSBs of the delay line value, and loops from 0 to 31 times before its output toggles to the value originally input to the block. With this approach, delay resolution of one tap over the full 255 tap operating range is achieved.

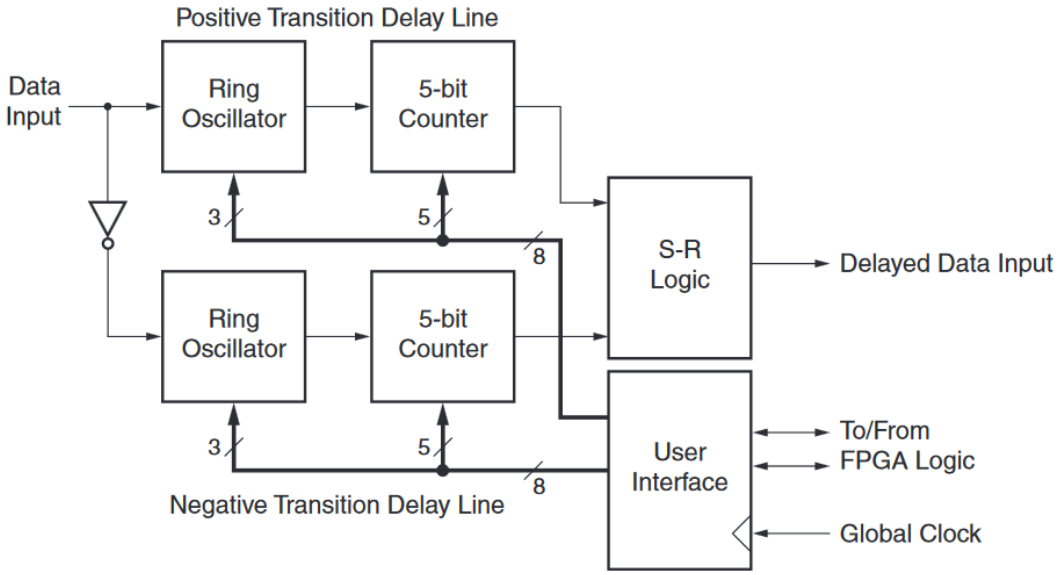


Figure 3.8: I/O delay block diagram in Spartan-6 FPGA [40]

This delay line has two major limitations. First, it can only be used to delay signals by a maximum of one bit period (2.5ns in ATG application), because bit errors might occur beyond this limit. Second, processing of an edge must be complete before another edge arrives. This can occur when received high-speed data streams having noticeable jitter. The first limitation is avoided by always keeping the delay less than the incoming bit period. Typically, the delay is set to 0.5 UI (1.25ns) to enable data sampling to be centered in the middle of the eye. In ATG, by controlling the trace length in advance, this constraint is not an issue since only fine adjustment (less than 500ps) should be needed. The second limitation is also avoided. Since each block actually contains two delay lines as shown in Figure 3.8, one delay line delays positive input transitions, and the other delay line delays negative input transitions. When adjusting delay values, both blocks change together to ensure positive and negative changes are delayed equally. The global clock running inside

FPGA is 200MHz, each delay step tap should perform ~20ps delay (5000ps/256 = 19.5ps). The actual delay performance is measured and shown in Figure 3.9 to Figure 3.11.

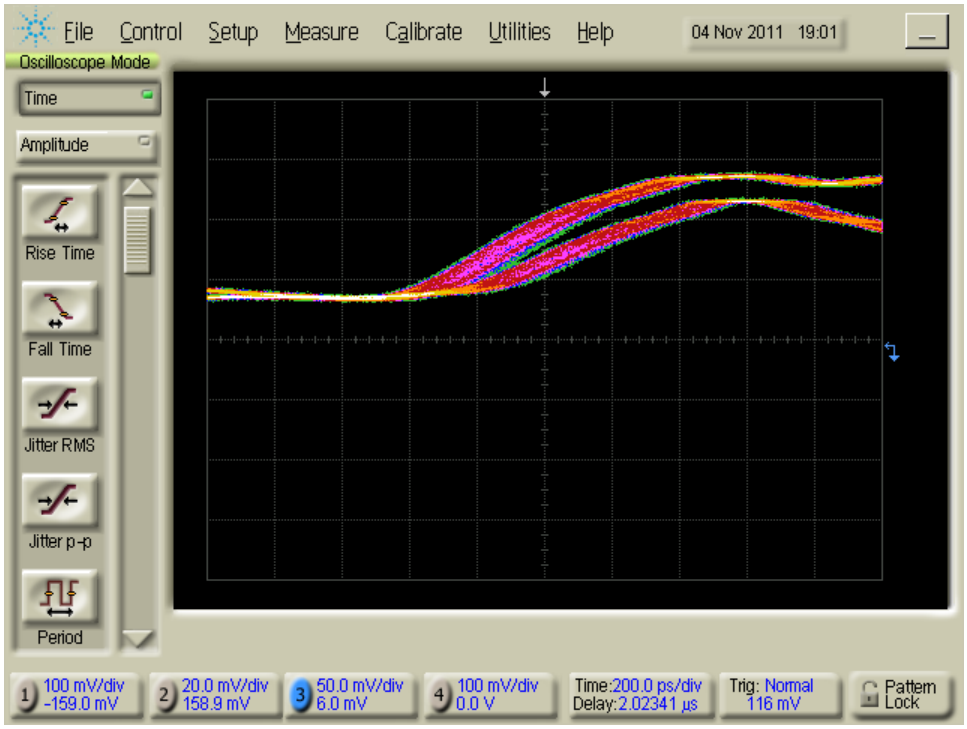


Figure 3.9: 10 Taps delay to create 200ps delay

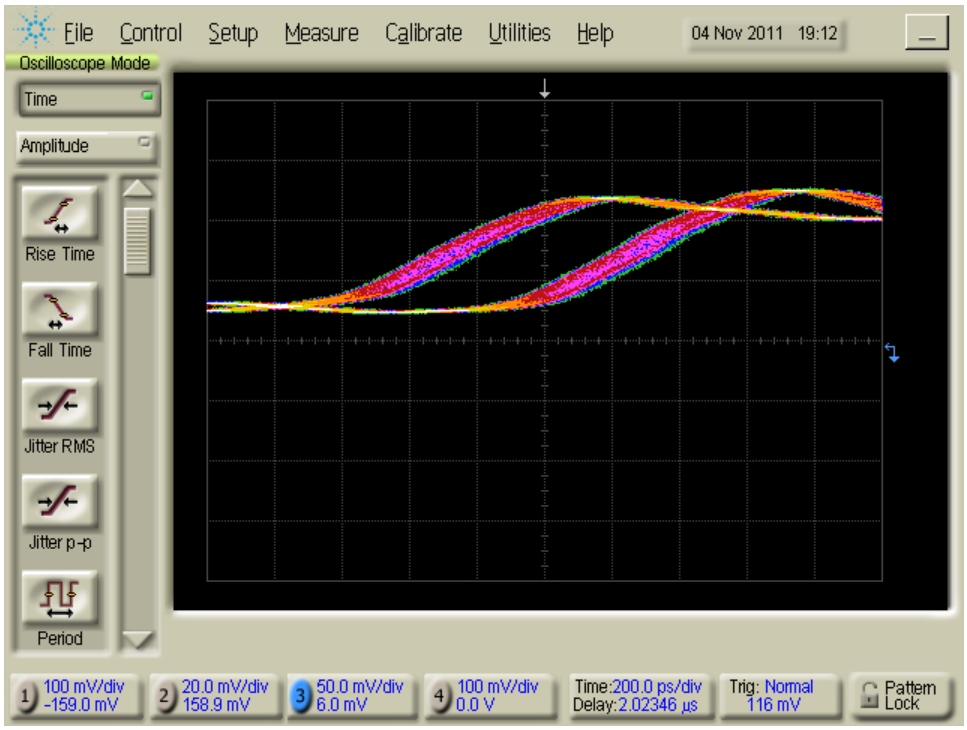


Figure 3.10: 20 Taps delay to create 400ps delay

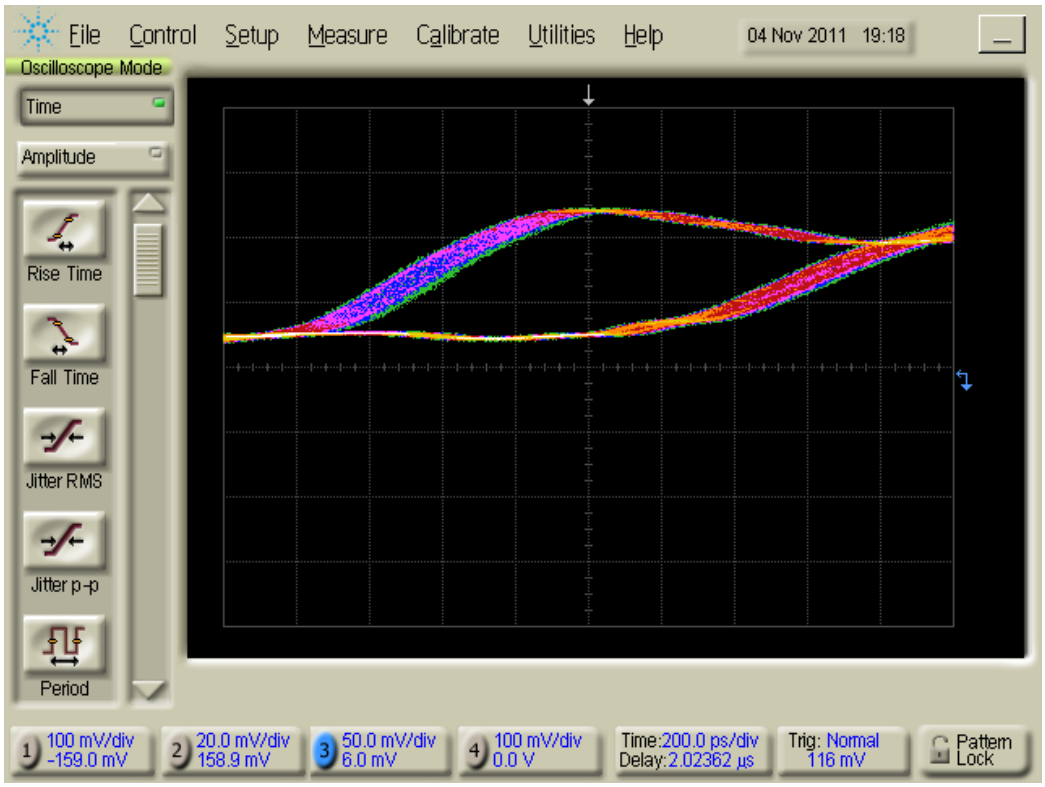


Figure 3.11: 40 Taps delay to create 1000ps delay

3.3.4 PCB Design and Stack up

The ATG board is built of an 8-layer PCB including 2 outer Rogers-4350 [42] layers and 5 inner FR4 layers, the target thickness is the common 62.5miles standard. The stack up of the PE board is shown in Figure 3.12. The thickness of 1 OZ Cu is about 1.35 mils, the outer signal layers are 4 mils and the inner signal/power layers are 8 or 10mils (adjusted to match total 62.5miles thickness). The top and bottom layer (Signal1 and Signal2) are high-speed signal layers since these two layer are built of low-loss Rogers material, the high-speed traces should be routed on these two layers to avoid signal loss. The transmission line on both two layer is Microstrip line [43], the high-speed traces are designed to be 8 mils [45] width based on the stack up and dielectric of the material to match 50 Ohms impedance to reduce the reflection and distortion. The signals on Inner1 are Stripeline [44] and are also routed to 8 miles width [45], the low-speed control traces

(analog pins, DC I/Os and SPI interface) are realized on this layer. The two power layers are both splitter power layers, multiple power planes are realized in each layer.

Figure 3.12: The stack up of 8-layer ATG board.

3.3.5 ATG System-level Design

The basic concept of ATG board is to use FPGA control core combines with outside-independent HS logics to produce the real-time delays and process the formatting and serialization of the multi-GHz signals [28]. Although FPGA itself is very powerful, there are several reasons we can't implement high-speed logics inside the FPGA [9] [10]. First of all, FPGA logic only runs at mid-hundred MHz, this speed is not able to build multiple-Gbps output. The other more critical reason is this "timing-on-the-fly" generator requires very precise timing control (~10ps), the resolution of the logic components inside FPGA is not enough (around 20ps which was illustrated in skew management section) for this purpose. Furthermore, we need to limit total jitter (that would otherwise be added by

the FPGA) to reach multi-GHz speed. Those high-speed components in separated HS section either provide enough bandwidth and resolution or minimize the total jitter of the timing-critical signals (clocks, in particular). Therefore, a more reasonable and achievable approach is to implement the complex ATG algorithm and heavy-load calculations within the FPGA, and use the calculated results from the algorithm to control those high-speed components outside the FPGA to generate the final waveform.

Since the detail ATG algorithm will be well-discussed in the next section, we will just briefly mention the big picture of the algorithm and focus more on ATG HSL operation here. First of all, a small digital control-word (fetch from the ATE system) signal is sent to the FPGA. Secondly, 16-bit of parallel "pattern data", and a compact "Timing Pointer" binary code is applied to the parallel data bus. This code is presented to the FPGA at rates up to 200 MegaWords/s (i.e. every 5ns). The "Timing Pointer" data is a compressed code that is used within the FPGA to address detailed timing values previously stored in memory. The detailed timing values are interpreted by the pipelined computation within the FPGA in order to calculate 16 sets of edge-generator timings (for all 8 edge generators) every 5ns. These 16 sets are interleaved within the FPGA using the DDR approach near the end of the pipeline, to produce data for all 8 generators every 2.5ns (400MWps). Each edge generator of HS logics gets 8-bit of binary delay information every 2.5ns. This is used to define the "fine" delay timing for the 8 edges that ultimately are used to produce the waveform timing for that 2.5ns period. The clock distributor generates eight delayed-reference clocks space at 312.5ps increments (0ps, 312.5ps, 625ps….2.1875ns), as Clk Phase1 to Clk Phase8 in Figure 3.13. The fine edge-generators take these delay reference clocks (Clk1 to Clk8) and produce 400MHz clocks with the desired phase-delay for each edge. The phases are updated every 2.5ns for each of the 8 generators. However, before the timing edges are combined (eventually by the XOR tree), the 8 timing edges are synthesized with decoded pattern data (decoding is done within the FPGA as part of the pipelined processing in order to account for the logic present in the XOR tree). Combining

the decoded pattern and timing data happens for each of the 8 edge generators in a single high-performance flip-flop ("FF1~8" in Figure 3.13), using the delayed version of a low-jitter 400MHz clock. The outputs of the 8 flip flops are later sent to an 8-input XOR tree where the logic transitions are effectively interleaved. This interleaving retains the edge-timing relationships of the 8 inputs without re-clocking. The low-jitter XOR logic decodes the encoded pattern data from total 8 FFs to produce the final output which is the desired multi-GHz serialized and formatted signal.

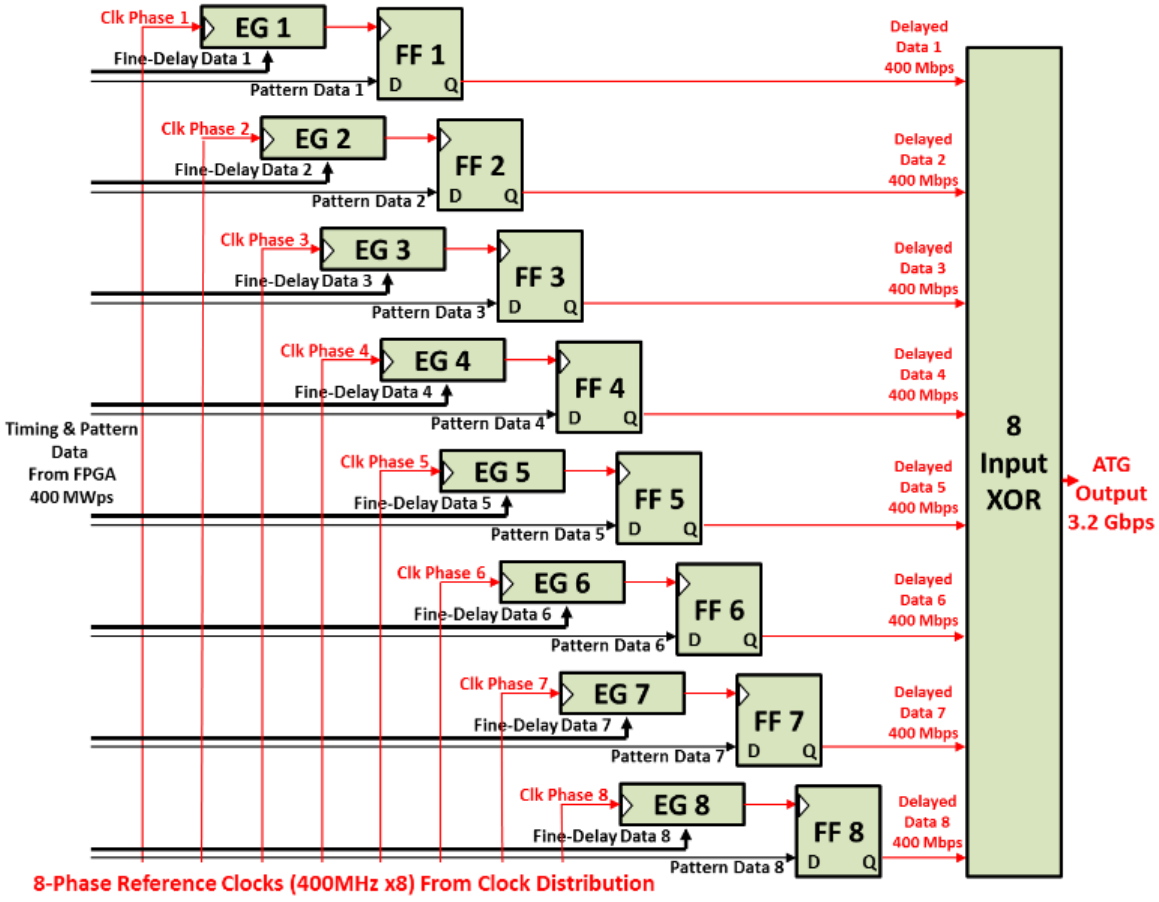


Figure 3.13: Internal features of the High-Speed Logic block.

The detailed (single-channel) ATG edge generator logic diagram is shown in Figure 3.14. One of the 8-phase reference clock (non-modulated with 50% duty cycle) is modulated by the fine delay value, afterward the delay-modulated signal is used as the CLOCK input to a high-speed register element (flip flop – "FF"). This delayed-modulated

clock is used to sample the input data of the FF. By using another digital serial bit stream (Pattern Data) from FPGA for the DATA input (D) to the flip flop, a delay-modulated serial data signal is produced on the flip flop output (Q).

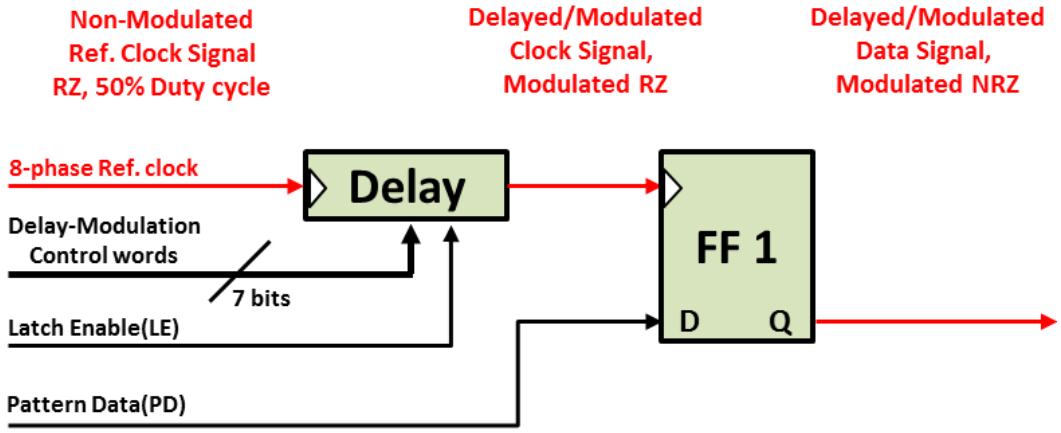


Figure 3.14: ATG Edge generator and flip-flop. Using a modulated clock signal to produce a modulated-NRZ data signal with timing-edge adjustability on a cycle-by-cycle basis.

The internal signal timing relationships within the ATG HS section are shown in Figure 3.15. This figure is helpful to get a better understanding of how the ATG creates the arbitrary multi-GHz signals. In part (A) of the figure, the timing of the 8 delayed-reference clocks are shown, at evenly-spaced 312.5ps increments, spanning one cycle at 400MHz (2.5ns). These are the 8 clocks that feed into the "Fine-Delay" edge generators shown in Figure 3.13. Each rising edge of these clocks represents one of the edge happening in final waveform. If we program equal "fine" delays for all 8 edges, then the entire output signal will be shifted by the amount illustrated as "skew" in part (B). This ability is generally needed in order to "deskew" each DUT I/O signal relative to the others (i.e. deskew multiple ATG outputs) or to introduce the desired phase relationships, and to realize modulated jitter injection. This "static" deskew ability is generally available on some high-end ATE today. However, we want to have more flexibility to create nearly unlimited timing waveforms where each edge delay can be independently adjusted, as illustrated in part (C) of the figure. We call this "dynamic" or "on-the-fly" skew of each edge. Part (D) and (E) shows the final output result of all signals from 8 phase generators XOR together.

The resulting ATG output signal would look like that shown in part (D) if a simple clocked data pattern (01010101) was programmed. The more important function is , we also allow any arbitrary pattern data or pseudo random binary sequence (PRBS) data to be programmed, so the resulting waveforms can be as shown in part (E) which represents the familiar data "eye" patterns. Keep in mind that each edge is programmed to within a 10ps resolution, and may be reprogrammed "on-the-fly" in any or all subsequent 2.5ns time periods. Therefore conclude all of above operations, we can construct a nearly-endless, unlimited sequence of these 2.5ns waveforms, the detail demonstration will be shown in the experimental results section later.

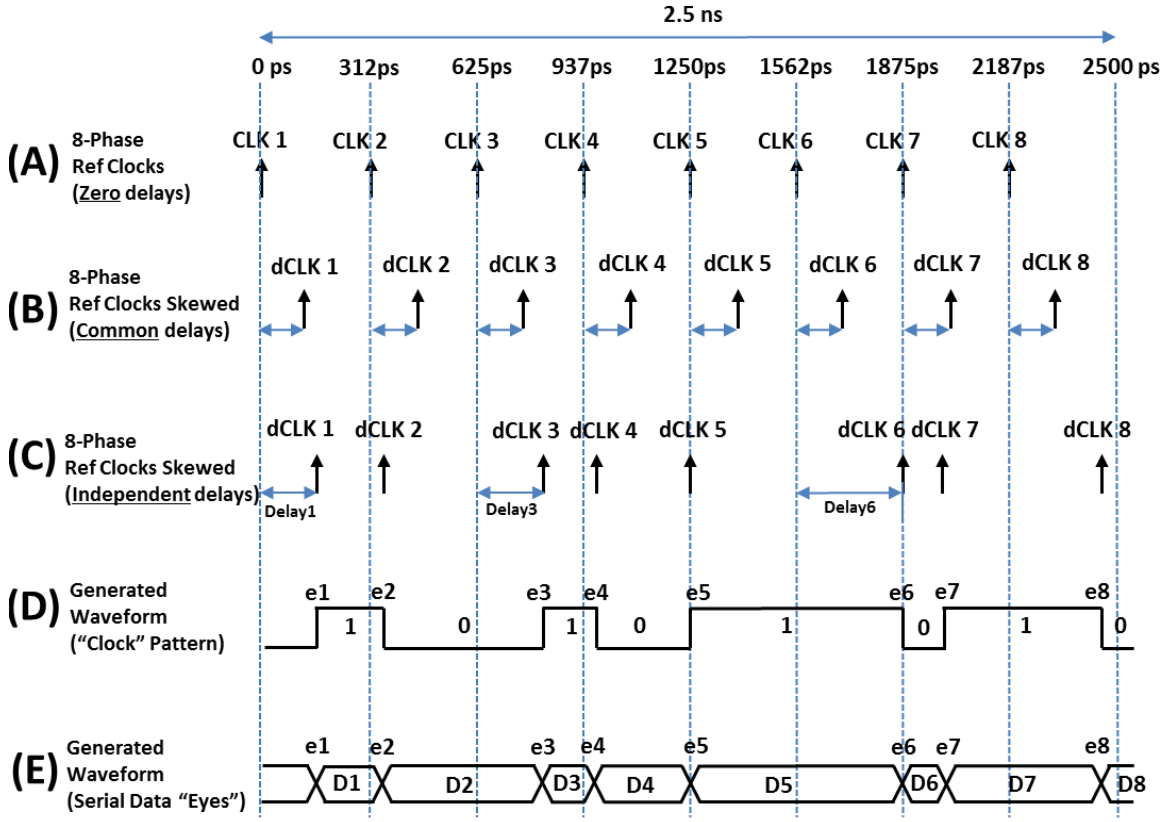


Figure 3.15: ATG timing relationships.

An example timing diagram of edge generator illustrate the relationship of the reference clock, controlled latch enable signal and pattern data is shown in Figure 3.16. Note that the resulting output bit stream is in a delayed-non-return-to-zero (DNRZ) format,

which is needed for the exclusive-OR multiplexing strategy that interleaves the 8 edges. This hardware design provides a potential solution of generating the desiring flexible waveform. However, the success of this ATG hardware requires very accurate control signals (fine delay value, latch enable and pattern data) for the HS section. These signals are generated by a real-time, high-throughput and reliable algorithm implemented inside the FPGA controller which will be discussed in the next section.

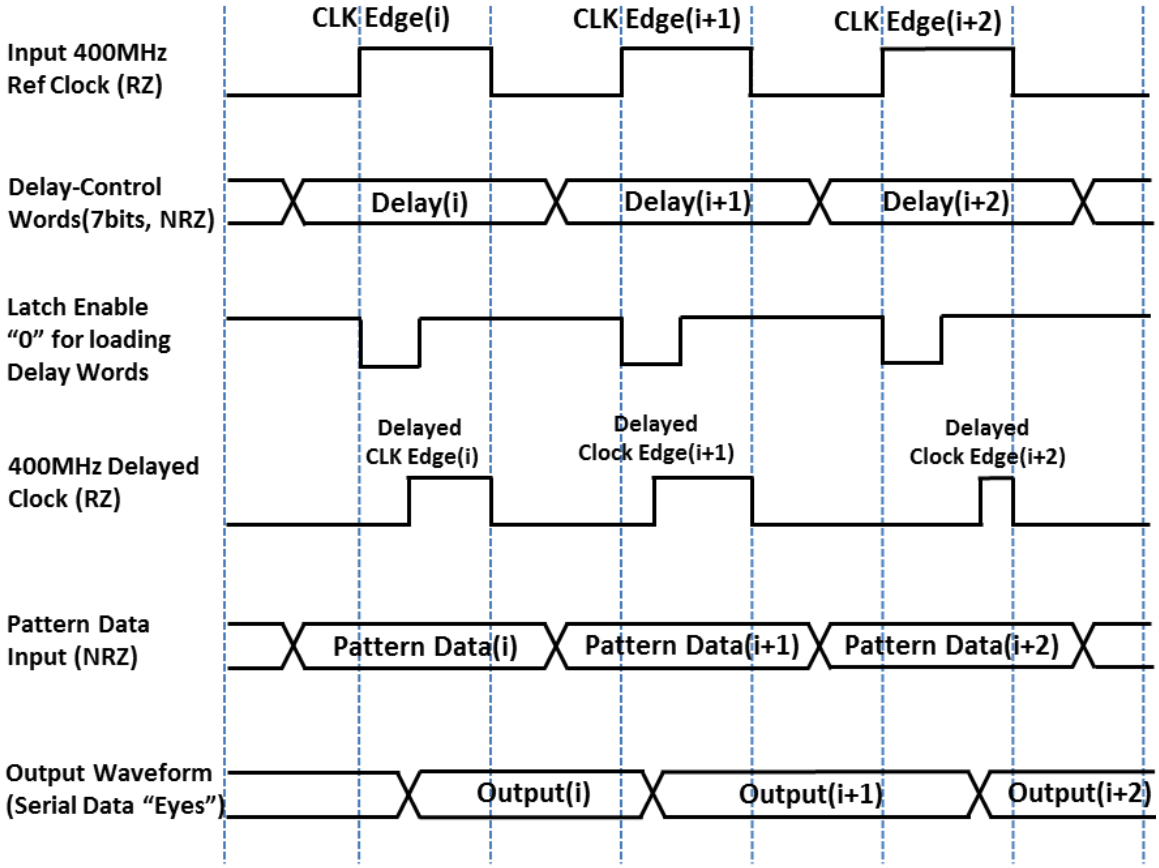


Figure 3.16: Example timing relationships in the extended ATG structure with pattern data.

3.4 FPGA Controller Design (ATG algorithm implementation)

Despite the implementation the hardware design, the more critical factor to make the system fully function is to generate the proper control signals of HS section in an appropriate sequence and at correct timing. Both value and arrival time of these control

signals in the HS section decides the success of the whole system. The detailed timing values are calculated using a combination of static memory (lookup tables) that store timing parametric values, and arithmetic logics decoded within the FPGA. The use of real-time algorithmic calculation provides the system nearly unlimited possibilities (with minimum constraints) for defining the desired timing signals. By the powerful arithmetic unit inside FPGA, it also reduces the number of external signals needed for controlling the ATG. Due to the heavy-loading parallel calculation requirement of the algorithm and the speed requirement of HS logics, the algorithm is accomplished by using a deeply-pipelined logic structure in order to match both FPGA operating frequency and HS section input data rate.

The algorithm implemented within FPGA (Figure 3.17) must process both the pattern data and the timing pointer data at 200MHz in order to generate control signals at 400MWps rate so that the HS section receives new control values every 2.5ns (for all 8 edge generators and the 8 flip-flops) from the output results of FPGA. The output timing values (Delay1~8) define the relative time that the edges will happen in the HS section with the resolution of 10ps. The decode process of timing value is done by edge decoder (upper part of Figure 3.17). The pattern decoder (lower part of Figure 3.17) decides the pattern of the data, which is "0" or "1". These output pattern (Pattern_out [1:8]) is actually defining rising or falling for the edges generated by edge generators. Furthermore, by implementation of pseudo random algorithm within FPGA, the system can easily generate PRBS data with different lengths (the demonstration in the experiment and result section shows the example of PRBS-31 pattern). The reference clock is distributed to FPGA banks by a global clock buffer at the center of the FPGA. This global buffer is very critical to synchronize the edge decoder, pattern decoder and the pipeline stages inside those decoder. It is unavoidable to introduce pipeline structure to this process, and more registers located in different banks within FPGA are used under this architecture. Therefore it causes unknown skew as the reference clock distributed to different banks of FPGA. The

implementation of global clock buffer will reduce the skew difference of the reference clock fanout to help synchronization.

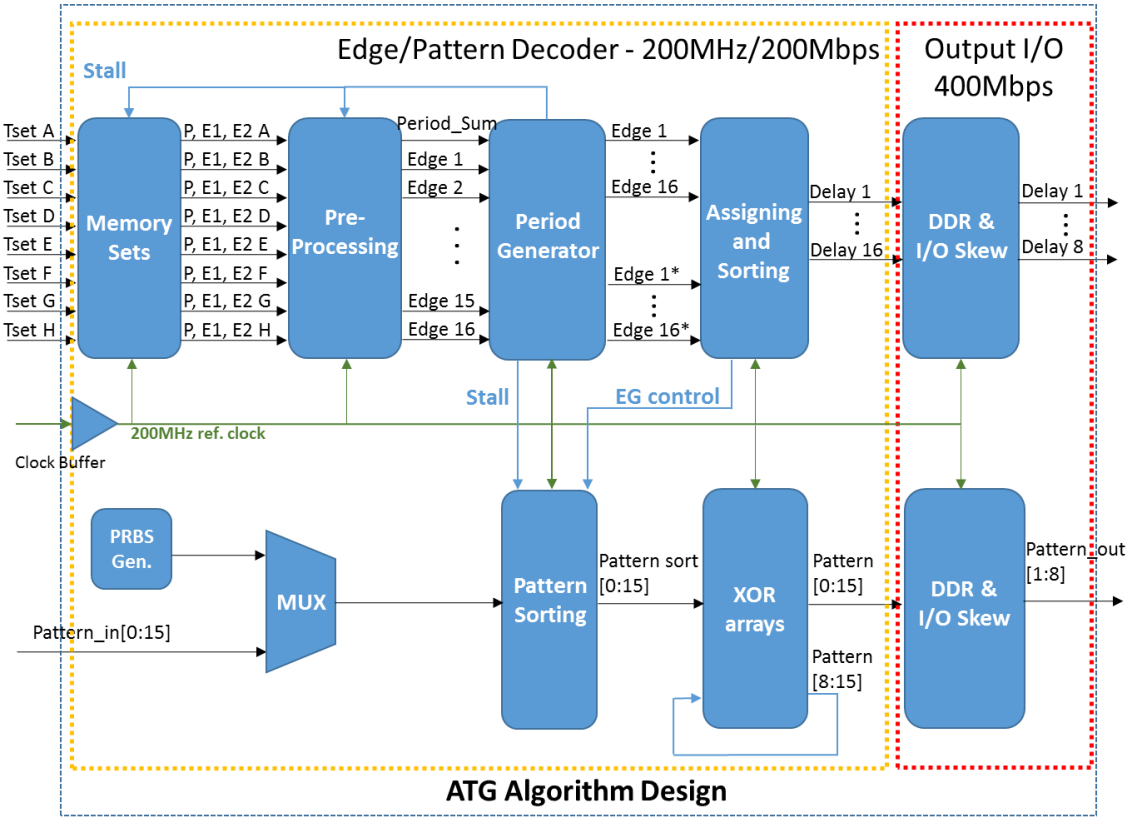


Figure 3.17: Top level of ATG algorithm in FPGA. The upper part is Edge decoder, and the lower part is Pattern decoder.

3.4.1 Edge decoder

Edge decoder decodes the timing information for edge generators in HS section, the edge decoder is divided into five major parts. In the first part, the input pointers fetch timing information stored in memory every 5ns (200MHz). This information is composed of one period and two edge information, as Figure 3.18 shows. Each memory unit stores a 69-bit wide word so that each period or edge information contains 23 bits. This 23-bit length word including 20 integer bits and 3 fractional bits, the least significant bit (LSB) of integer part is 10ps, therefore this format is able to represent timing from 1.25ps (LSB of fractional part) to 20ms. This format allows the algorithm to generate both wide band and high-resolution control signals for the HS section. The use of memory sets also helps

to reduce the signal wires in the system. If the memory is not implemented, around 552 (69 bits, 8 sets) wires are required to achieve the similar function. The other advantage of using memory to store these information is the flexibility. If digital designers require changing timing information, or in other word, to form different waveform for different testing scenarios, only a simple re-configuration to the timing value stored in memory is needed. So far the FPGA just simply fetch timing information out of memory, the main algorithm of the following four parts in edge decoder will be detailed in the rest of this section.

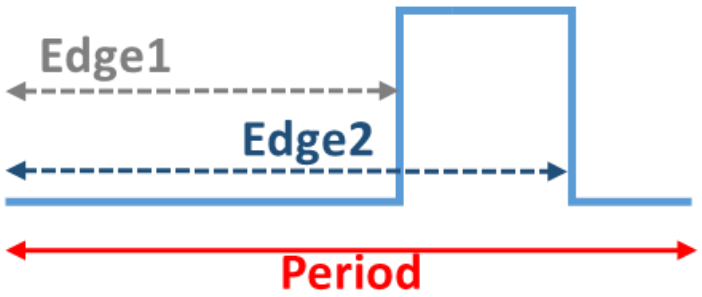


Figure 3.18: Timing-value format in memory. Each memory set includes one period and two edge information (in binary form).

3.4.1.1 Pre-processing

Each timing set contains one period and two edges is stored within a memory address. However, the waveform we want to generate is continuous. Therefore in Pre-Processing section, the individual and separated timing information of intended waveform sets will be calculated and transfer to a continuous form. Since there are 8 periods and 16 edges needed to be calculated in this section, and also the wide bit-width of each calculation (including integer and fraction part). The operation loading of this section is very heavy and a pipeline structure is required to complete each calculation in 5ns period. The flowchart of this process is shown in Figure 3.19, and it can also be generally presented as a mathematical form as Equation 3.1. This formula describes the continuous edge value is the summation of all earlier period values and current edge value.

$$Edge_{sumi} = (\sum_{i=1}^{i} Period_{i-1}) + Edge_i \qquad \text{Equation 3.1}$$

Furthermore, the summation of all 8 period information also needs to be calculated in this stage, the formula is shown in Equation 3.2. This information will be used for cycle count and determining the appropriate cycle for these edges in the next stage.

$$Period\_total = \sum_{i=1}^{8} Period_i \qquad \text{Equation 3.2}$$

Each pipeline stage contains numbers of mathematical operations, and each operation is built by a >24-bit fractional adder (increased in later stage) which is composed of >21-bit integer part and 3-bit fractional part. These adders are actually transferring the timing information in each parallel and individual waveform set into a serial and continuous format, the total bits of the continuous edge and period sum information needs to be expend to fit the calculated results. A simplified example of this process is shown in Figure 3.20, which adds four individual waveform to become a serial waveform.

Figure 3.19: The block diagram of Pre-Processing stage

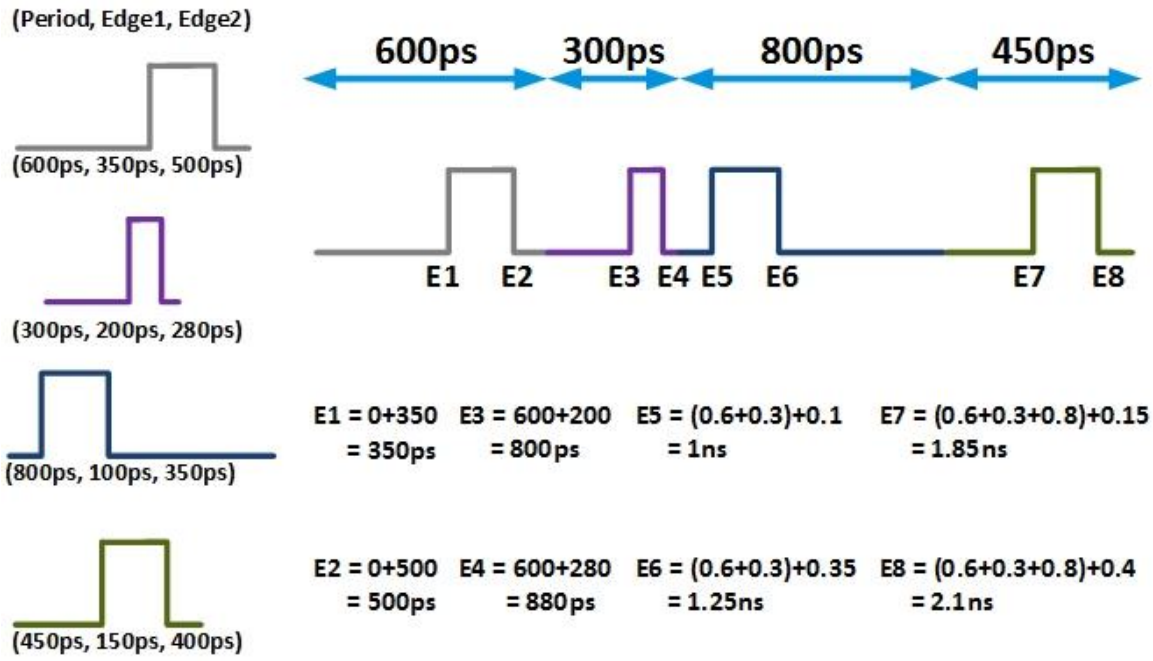


Figure 3.20: Example of adding several discrete edges to form continuous signal.

3.4.1.2 Period Generator

Period Generator (PG) uses the continuous edge and period sum information from Pre-Processing stage to generate output cycle and assign the edge values to appropriate output cycle. In other words, PG actually transfers the edge values from FPGA operating domain (fetch cycle) to HS operating domain (output cycle). Since the FPGA is running at 200MHz, the typical period of fetch cycle is 5ns. If those edges are happening within 5ns, then those data can be kept in the current cycle, which means the output cycle is synchronized to fetch cycle. However if some edges or total period exceed 5ns, it will be assigned to next output cycle therefore output cycle is not synchronized to fetch cycle. The out of sync usually happens as a lower data rate output (<3.2Gbps) is presented. Figure 3.21 shows the process of PG, a three-stage pipeline first takes period sum to determine the total output cycles required, and this process will keep calculating to generate new output cycles. Each edge (only one is shown in Figure 3.21) is specified respectively if it is in current output cycle. If it exceeds 5ns, then it will be held until the correct output cycle

is generated. As the correct output cycle comes, these residue edges from earlier fetch cycle are sent to stage 7. Finally the residue edges and edges in current fetch cycle will be integrated into the same output cycle and send into stage 8 for next process.

Furthermore, the delay offset values for edge generators in HS section are calculated in this section. By the process of determining the appropriate output cycle, the edge values will be subtracted to less than 5ns. The new edge values are used to control and generate delayed waveform in edge generators. An important control signal called "stall" is generated in PG to freeze or continue the process of the system. Stall signal is very critical since it guarantees the success of switching different frequencies. If this control signal is not generated and sent appropriately, for example, it does not stall the operation of the pipeline stages at correct cycle, we might loss the edge information we want to generate and the timing dead zone will occur in the final result. This situation usually happens when we want to switch to a lower-speed (under 1Gbps) signal. The total period of low-speed signal is longer than 5ns, which means the edges are not synchronized to fetch cycle and need to be held until the correct output cycle. Remember FPGA will continue to fetch data from memory sets at 200MHz frequency, so the period and edge information stored in the register will be refreshed by new values every 5ns. To ensure this system to operate at any frequency with "timing-on-the-fly" ability, this stall signal is required to be generated accurately. Once the system detects period sum is longer than 5ns, the stall signal is asserted to stop the fetching and calculating process. In the meantime, the period sum keep subtracting 5ns to get new period sum and generate next output cycle, then the new period sum will be tested again until it is less than 5ns. Once the new period sum is less than 5ns, then stall signal is de-asserted and sets fetching and calculation back to normal operation. The other issue of generating this signal is it needs to be fanout to all the registers in PG, Pre-processing stage and memory set. According to the previous stage, there are over 10,000 registers need to be stall simultaneously, that also means over 10,000 fanout are required to spread this signal to all the registers needed to stall. More fanout

increases the transition time and delays stall signal transmission. The delay of stall signal transition might freeze the registers after the value is refreshed, which causes errors happen (or bit loss) in the algorithm. In order to deliver this stall signal to those registers within a cycle time (5ns), an appropriate buffer structure design is required to optimize the stall signal transition time within 5ns.

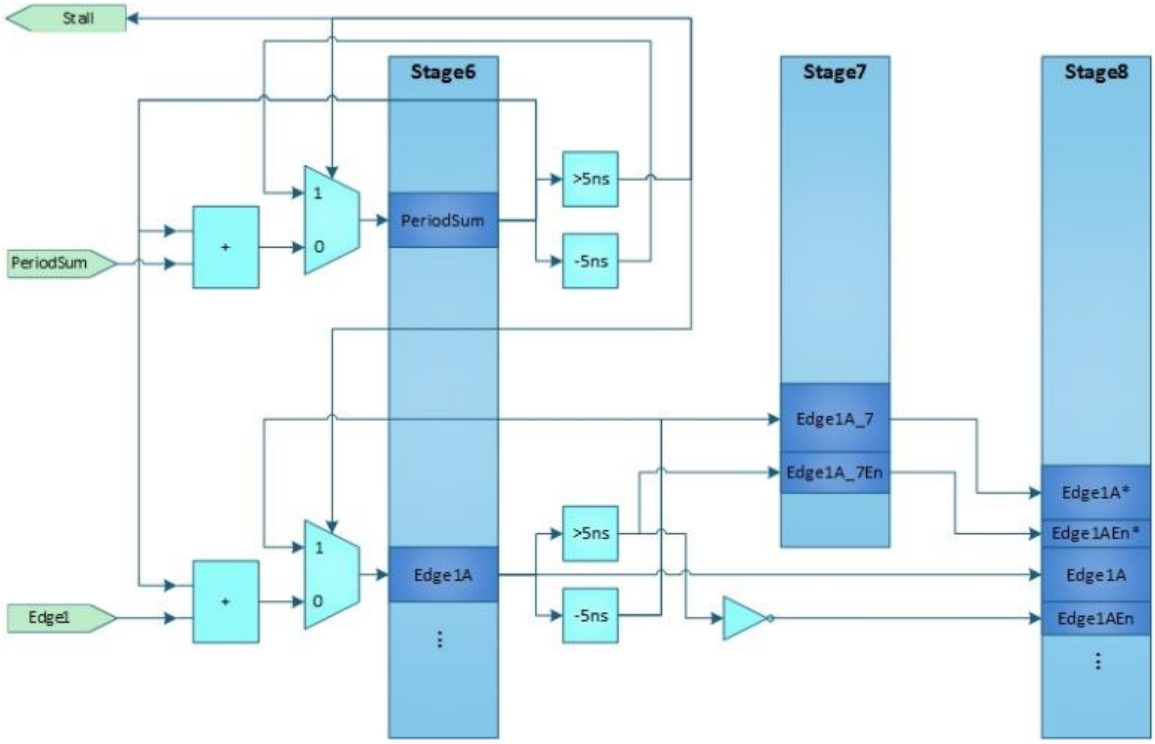


Figure 3.21: The block diagram of Period Generator algorithm

3.4.1.3 Assigning & Sorting algorithm

The main function of Assigning & Sorting (A&S) section is to reformat the calculated edge-timing values and assign these values to specific edge generators in HSL, simultaneously the algorithm will try to optimize those edge generators slot therefore we can use as many timing slots as possible to avoid dead-zone issue. The first step of A&S section is cropping. From PG stage, 32 edge (Edge1~16, Edge1*~Edge16*) slots are prepared for the edge value storage. However there are only 16 generators in 5ns output period (8 generators in 2.5ns period), therefore the empty edge slots need to be eliminated.

Since the maximum data rate of ATG system is to output 3.2Gbps signal (normal mode), which also means 16 edges happen in 5ns, the maximum cropping edge slots should be 16. The eight edge generators are used twice to cover total 5ns delay range, each edge generator handles 937.5ps delay range and every generator is 312.5ps behind earlier one, shown in Figure 3.22. Also two generators (Delay-1, Delay-2) from previous output cycle (residue of Delay7 and Delay8) can be used in current output cycle. Therefore we should crop 18 total edge slots for assigning process. The cropping 32 to 18 MUX is implemented between stage 8 and stage 9 in Figure 3.23.

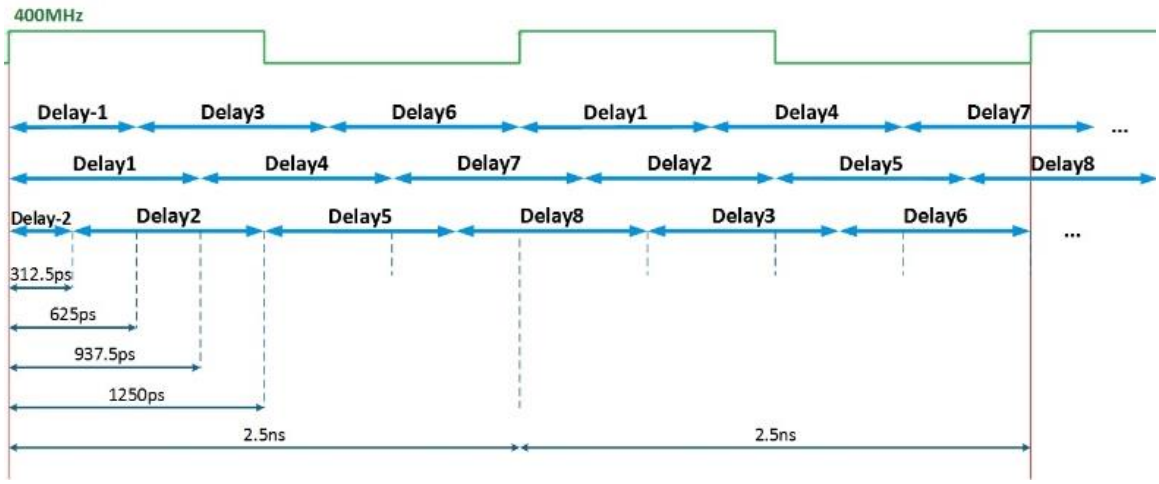


Figure 3.22: Usable edge generators in 5ns period. Each generator handles 937.5ps and is 312.5ps behind earlier one.

Figure 3.23 also shows the condition-branch algorithm for implementing edge-assignment. In the stage 9, 16-case condition test is used to verify both the timing range and if the edge generator is occupied. Every cropped edge data from the output of cropping MUX is sent into an initial edge generator respectively, then the algorithm checks the timing value if it is in the appropriate edge generator. According to Figure 3.22, any edge can be processed by three different time generators. For instance, edge value 800ps can be processed by Delay1 (0~937.5ps), Delay2 (312.5~1250ps) and Delay3 (625~1562.5ps). The algorithm in the meantime verifies if any of the three generator is used by earlier edges and store the time value to the earliest and unoccupied generator among the three

generators. Back to the example above, if the Delay1 is used by an edge value of 500ps, then the 800ps edge value will be sent to Delay2. Since this algorithm is implemented by a large branch logic, it is impossible for the whole calculation to be completed within 5ns FPGA operation cycle. Therefore an 18-stage pipeline structure is introduced to assigning algorithm (in each pipeline stage only one edge value will complete the assignment) in order to make the calculation in each stage happens within one FPGA period. In this section, there exists similar issue like PG stage, the fanout of the giant branch logic might slow the speed of the calculation down in this stage. Fortunately this deep-pipelined structure is able to help reduce the branches and release loading of fanout significantly.

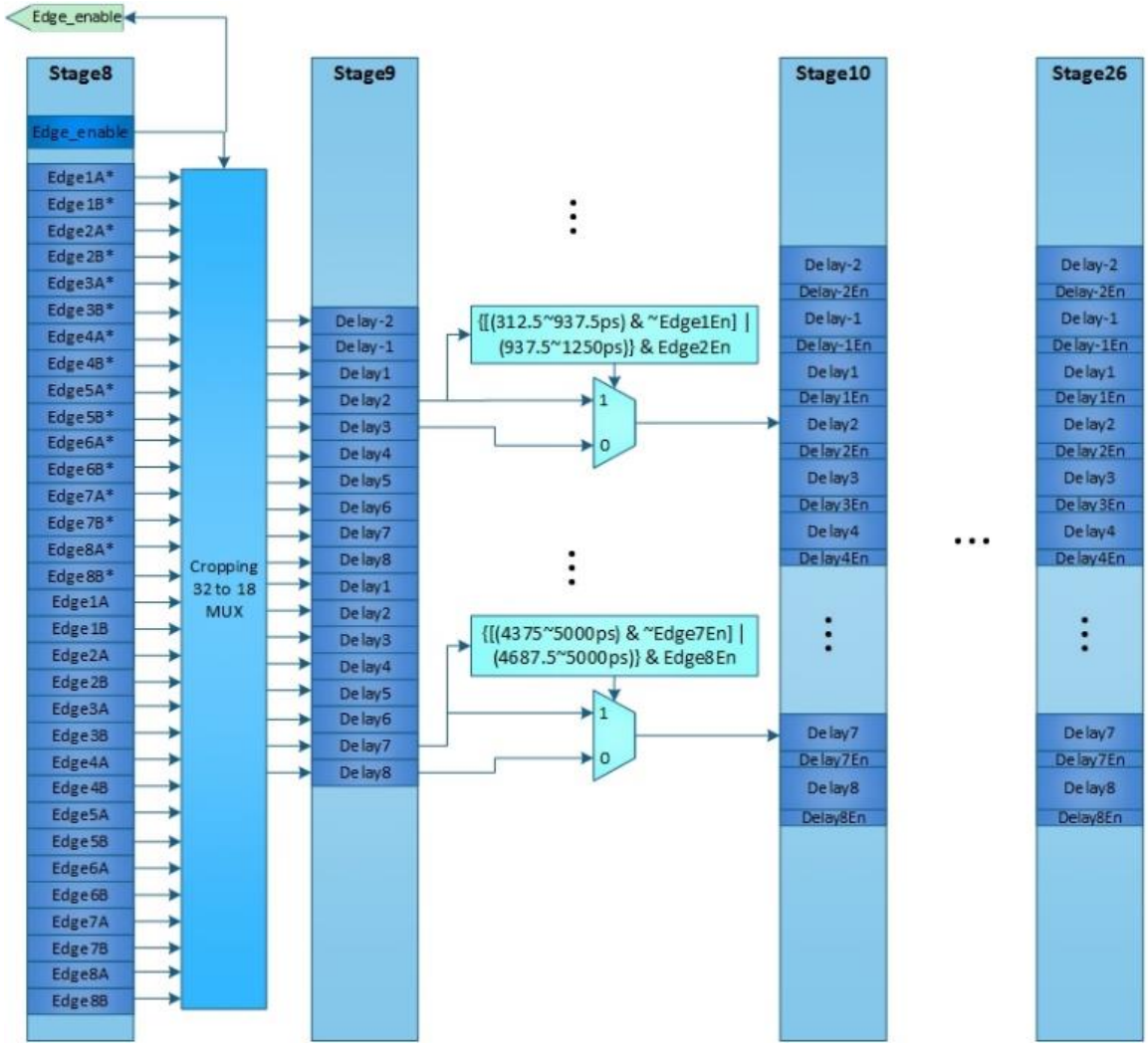


Figure 3.23: The block diagram of Cropping and Assigning algorithm.

In the assigning step, 18 edge values were sent into the appropriate edge generator (which are Delay-2, Delay-1, two sets of Delay1~8). However, Delay-2 and Delay-1 are not real generators in current output cycle. We know only eight physical edge generators exist in the HS section, therefore Delay-2 and Delay-1 are actually the residue timing ranges of Delay7 and Delay8 of previous output cycle. The final sorting process in this section is to send the edge values within dummy generators Delay-2 and Delay-1 back to real generators Delay7 or Delay8 if they are not occupied, otherwise they will be sent into Delay1 and Delay2 in current output cycle. The process is shown in Figure 3.24.

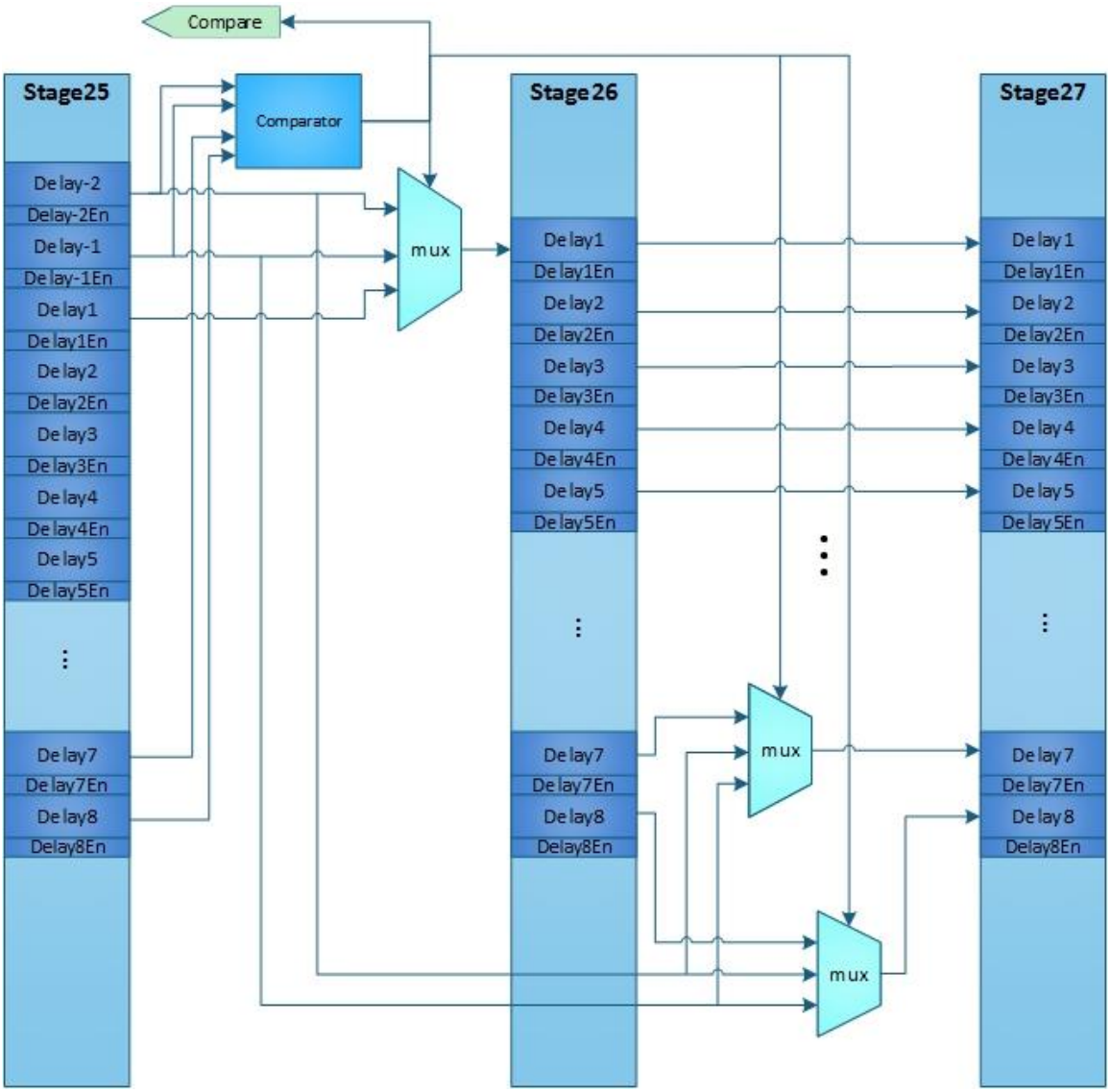


Figure 3.24: The block diagram of Sorting Algorithm.

3.4.1.4 DDR registers and Delay I/O

The final stage of the algorithm is composed of double date rate (DDR) registers and I/O skew management. So far we got sixteen 200MWps (FPGA frequency) time values (7-bit word width) for the control in HS section. However the HS section outside FPGA is running at 400MHz and the control of time values need to be refreshed every 2.5ns, also there are only eight edge generators implemented to format the arbitrary waveform. Therefore in final stage we need to double the data rate from 200MWps to 400MWps to match the HS operating frequency. DDR is the technology which samples the data at both rising and falling edge of reference clock to double the original data rate, a simple timing chart [40] is shown in Figure 3.25. In the ATG algorithm, DDR will sample the first eight edges (Delay1~8) at rising edge and the last eight edges (Delay9~16) at falling edge and output final 8-channel timing values at 400MWps for HS section.

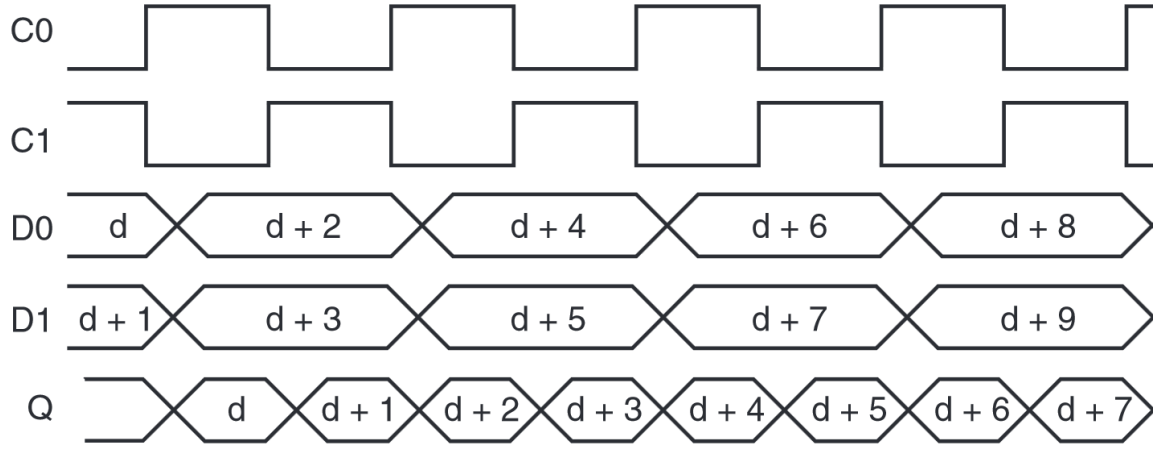


Figure 3.25: DDR timing chart.

One of the possible issues might fail this system is the phase skew issue between the seven bits in each time values. This problem and the solution is well-discussed in chapter 3.3.3. In the FPGA operation, the values of delay tap is programmed to ‘0’ in initial state. If the edge located at incorrect time, we can determine that certain of delay values are not captured correctly. Then we measure and calculate a new delay tap and programmed into the skew management. This process continues until all the edges occur at right time.

Furthermore, in this section the eight 400Mbps latch enable signals are generated to enable/disable edge generators. These signals require to be de-asserted ahead of un-delayed reference clock arriving and asserted right after the current delayed edge is generated, which is shown in Figure 3.3 and is well-illustrated in chapter 3.3.1. To make all these process to function every 2.5ns, the skew adjustment on latch enable signal is as critical as timing values.

3.4.2 Pattern decoder

The pattern decoder defines data pattern of the desire waveform, the data pattern in ATG system also can be re-configured by users. In Edge decoder, we decide the timing that the edges happen, and in Pattern decoder we define data pattern of those edges (0 or 1). Each data pattern is paired to a single edge value, therefore total 16 data pattern sets are fetched each cycle. FPGA can take these pattern data from ATE, memory or built-in PRBS generator then follows the similar process of edge decoder. This pattern decoder doesn't perform heavy timing value calculations as Edge decoder. It only assigns and generates formatted (decoded) "pattern-data" used as data input(s) to the timing generation flip-flops (TGFF) in HS section. The data needs to pass the same number of pipeline stages as edge decoder to synchronize with edge values. The initial pattern data will be processed by the deep-pipelined algorithm composed of Pattern Processing and XOR array to generate final pattern sequence to control TGFFs.

3.4.2.1 Pattern Processing

The main goal of Pattern Processing (PP) is to synchronize the pattern data with the paired edge value at correct output cycle and assign it to the appropriate TGFF. This process is similar to role of PG and Assigning & Sorting in Edge decoder. Each pattern data is linked to an edge value, the first part of PP performs the cycle determination and the process is also controlled by the stall signal generated from PG. Furthermore, the edge

enable signals of PG help to pair and synchronize the data pattern with the related edge timing, the process is shown in Figure 3.26.

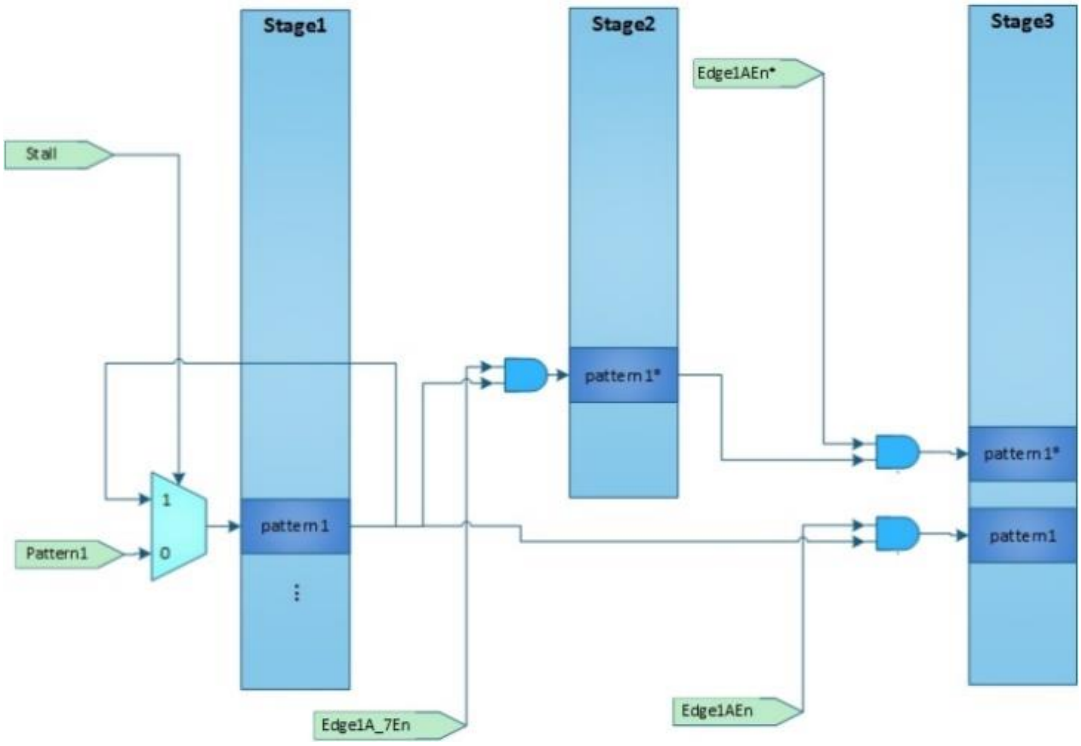


Figure 3.26: Pattern Processing: cycle determination.

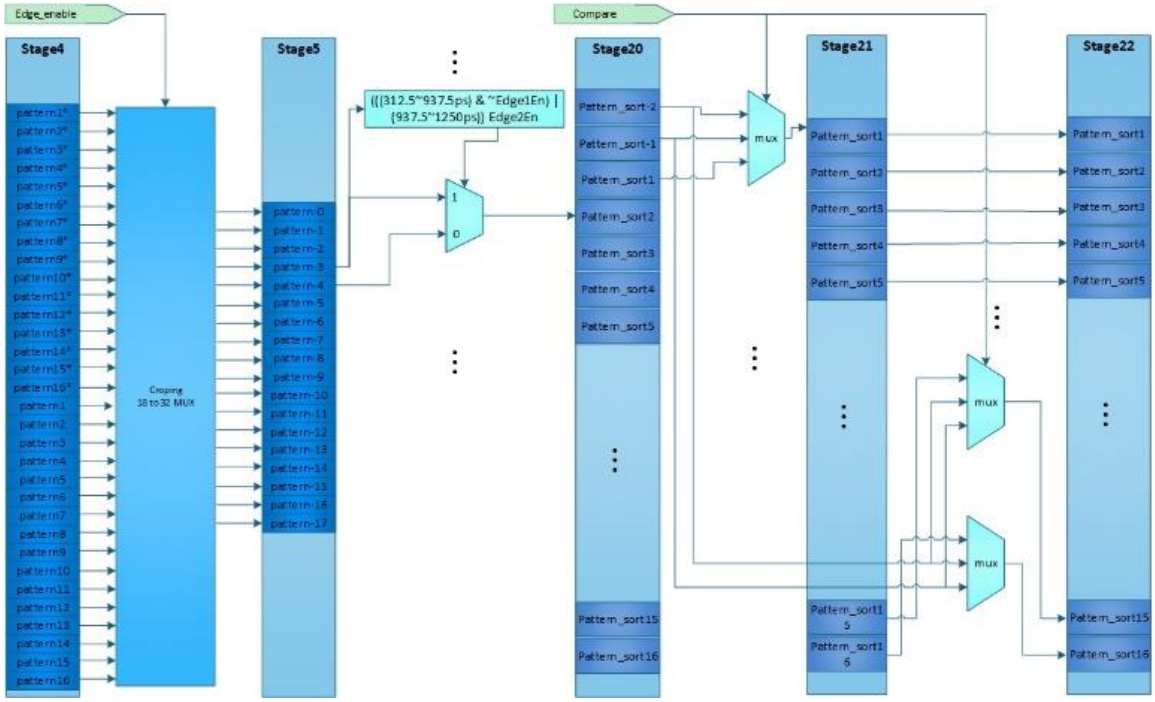


Figure 3.27: Pattern Processing: FF determination.

The second part of PP is shown in Figure 3.27. In this part, the appropriate TGFF is decided by the similar process of determining edge generator. Since each TGFF is paired to an edge generator in HS section, PP uses the edge values and edge enable signals from edge decoder to determine the appropriate TGFF as well.

3.4.2.2 XOR array

In this stage, the data pattern from PP stage is calculated through a XOR-based algorithm to achieve the final data and transmit into TGFFs. Since the output of each TGFF is toggling on rising-edge of a delayed and continuous clock from edge generator, the current data pattern is highly related to the previous data patterns. Also, the outputs of eight TGFFs are serialized to a high-speed signal, so the pattern of each late edge is associated with the early edges in current cycle. Therefore the original data pattern needs to be processed to fit the operation of TGFFs. From the PP stage we get 16 bits data D [16:1] in each cycle. However, we know that there are only 8 TGFFs in HS section so the data rate will be doubled in the next stage, therefore it is obvious that the 8 LSB bits (D [8:1]) are for the first output cycle and 8 MSB bits (D [16:9]) are for the second output cycle. In order to calculate the cycle-to-cycle related pattern (C [16:1]) to for the edges, a large XOR-array algorithm is developed, which the general formula (Equation 3.3) is shown below:

$$C[i] = D[i] \oplus C[i-1] \oplus \dots C[i-6] \quad i = 1, 2, \dots 16$$
$$C[i] = C^{-1}[i+16] \quad i \le 0, \quad C^{-1}[i]: previous\ cycle \qquad \text{Equation 3.3}$$

Since the MSB pattern calculation requires the output result of LSB pattern calculation, a two-stage pipeline is implemented and each stage includes a XOR array calculation, shown in Figure 3.28. Therefore total two FPGA cycles are required to get all 16 bits final data pattern accomplished. The calculation result of first half is held and stored within registers for the calculation of the other half. The previous cycle of the first output cycle is set to “00000000” at initial state (no edge happens before the first cycle). After this

XOR array calculation, these data will be sent to DDR registers and IO delay component to double the data rate and de-skew the signal phase to match HS timing.

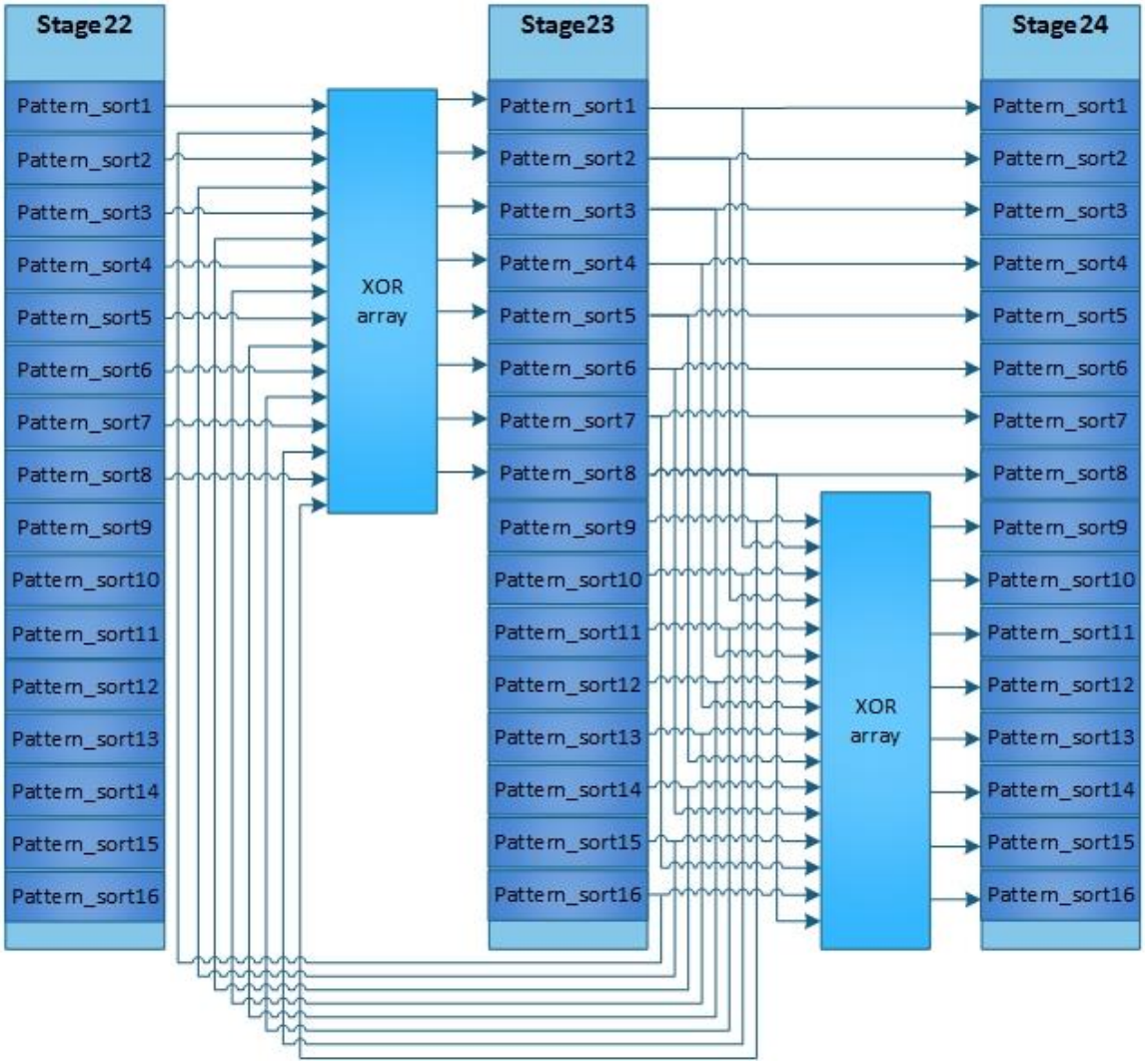


Figure 3.28: XOR array stage.

3.4.3 ATG algorithm operation

Since the discussion of algorithm and math may still be vague for a full understanding the whole process, a real example is used in this section to give a clearer picture of the algorithm. In order to simplify this example and also because of the high correspondence between Pattern decoder and Edge decoder, we will just use Edge decoder for explanation here and assume a clocked pattern (0101…) is used. In Figure 3.29, we

assume there are four different types of basic time sets (Tset 0~3) stored in FPGA memory to build arbitrary waveform. In each FPGA cycle, the FPGA can fetch maximum eight timing sets (or less) for serializing. Total 23 timing sets from three fetching cycles (represented by the depth of color blue) in this example are giving the desired waveform in Figure 3.29. In order to show the capability of the algorithm, we mandatory program the total period of first fetch cycle to exceed 5ns and only fetch 7 timing sets in the last fetch cycle. To format this waveform, a flow chart shown in Figure 3.30 is used to help readers understand the process. First of all, to clarify the FPGA fetch and output cycle (or HS cycle), we again use the depth of color blue to represent the FPGA fetch cycle, and we use different columns for different output cycles. Figure 3.30 basically shows the process after the calculation in Pre-Processing section (which fetch cycle and output cycle are still synchronized). The PG transfers all the edge information to output cycle domain, and the following three-step Assigning & Sorting function assigns these edge values to the appropriate edge generator in output domain.

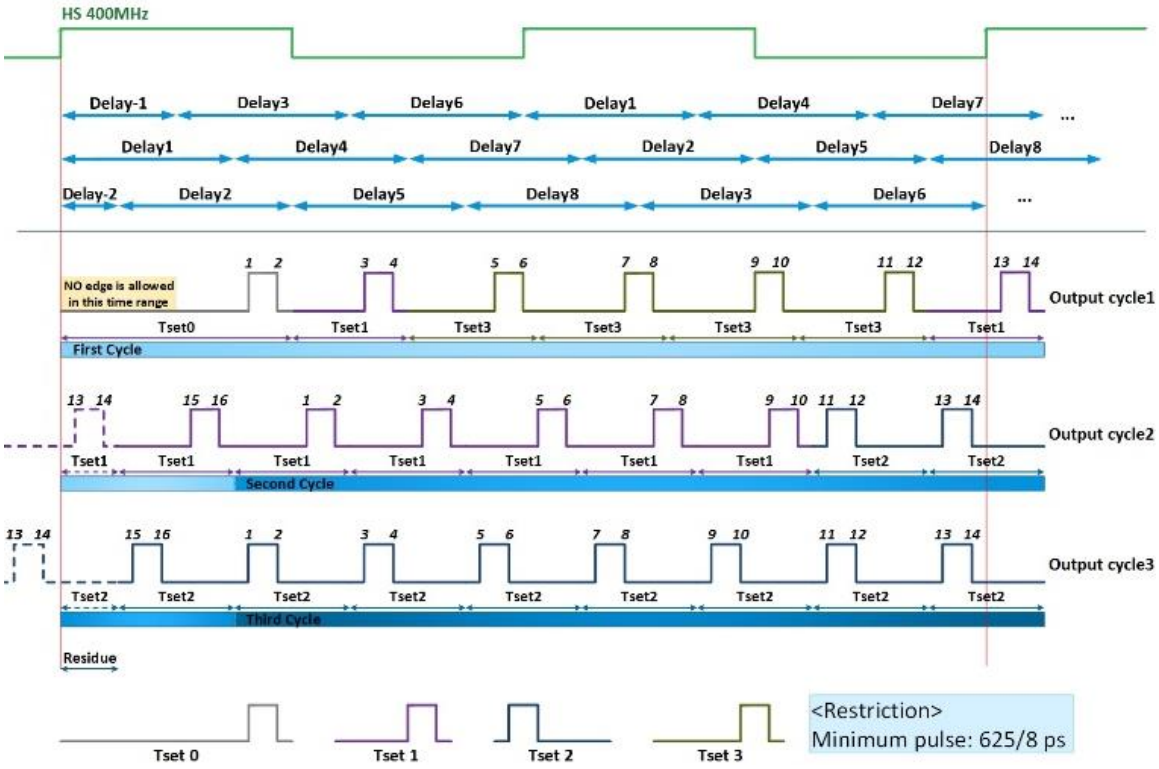


Figure 3.29: An example of assigning timing value to the correct edge generator.

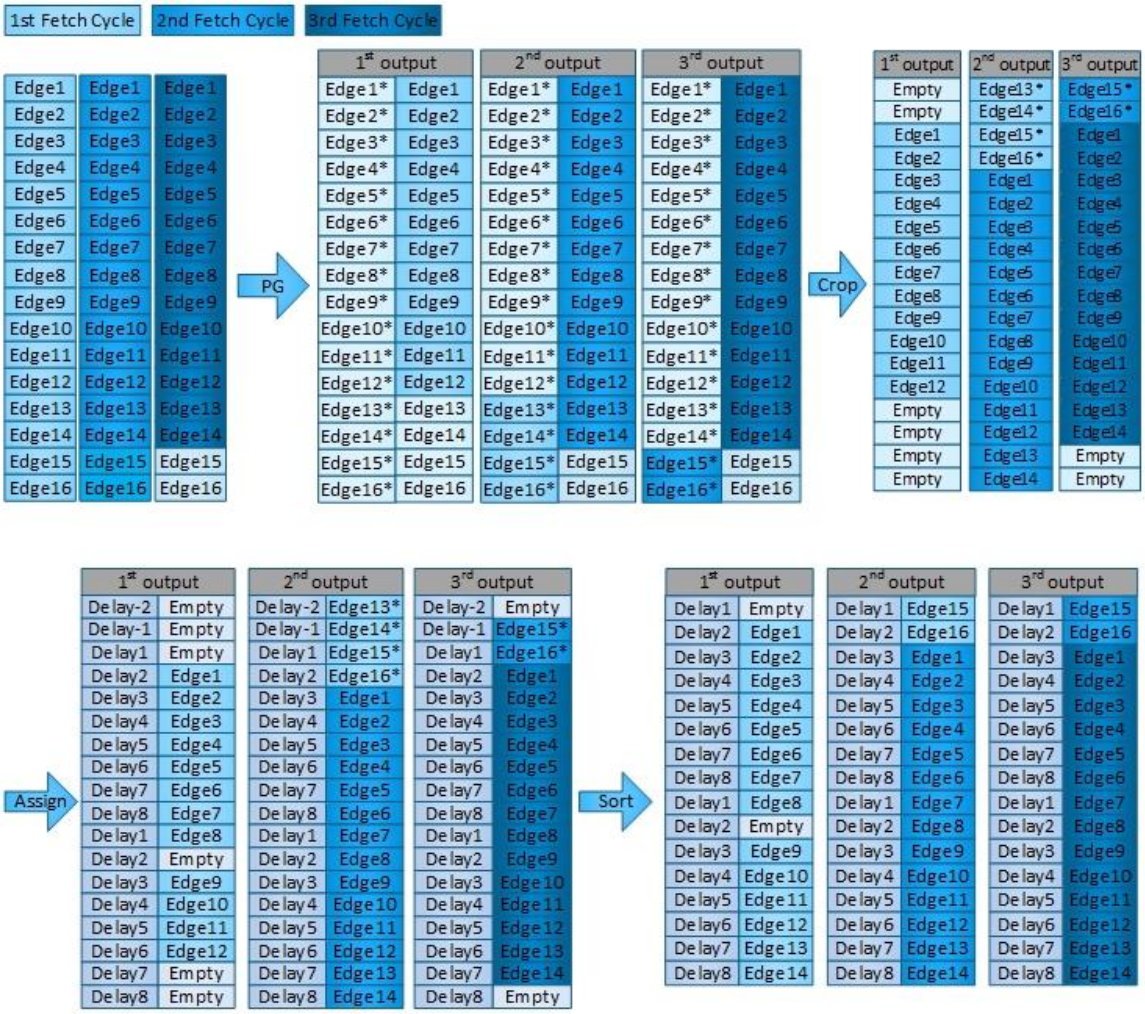


Figure 3.30: Example data-flow of algorithm for ATG.

In short, the ATG algorithm acts as the most critical role of controlling the ATG to perform the continuous timing-on-the-fly ability. Without this complex algorithm inside FPGA, the HS section itself can only generate several fix data rates (such as multiple rate of reference clock), whole phase shift (not individual edge shift) and produce waveform dead zone (as frequency sweeps) just like most of traditional ATE systems. Therefore the implementation of the algorithm provides this ATG extension module full capability of realizing wideband (DC to 3.2Gbps), continuous (no dead zone) and timing-on-the-fly (individual edge shift) to enhance the functions of current ATE systems.

3.5 ATG Hardware Performance and Testing Result

The ATG was constructed using standard PCB technology with "off-the-shelf" components, shown in Figure 3.31. The Xilinx Spartan-6 FPGA which the ATG algorithm is implemented located in the center of the board and connected to all the HS components. In order to minimize the phase skew and reduce the distortion of the signal, the trace is designed to both length and impedance match. The color wires are JTAG connectors which is used to program FPGA. The bottom side is a Tyco connector used as the interface of ATG board and ATE. The power supply can be offered by either ATE system or external supplies. Also 200MHz reference clock for FPGA and 400MHz clock for HS section are designed to be provided by either ATE or external source. On the top is the 26GHz-bandwidth SMA connector to output high-speed and serialized signal generated by ATG.

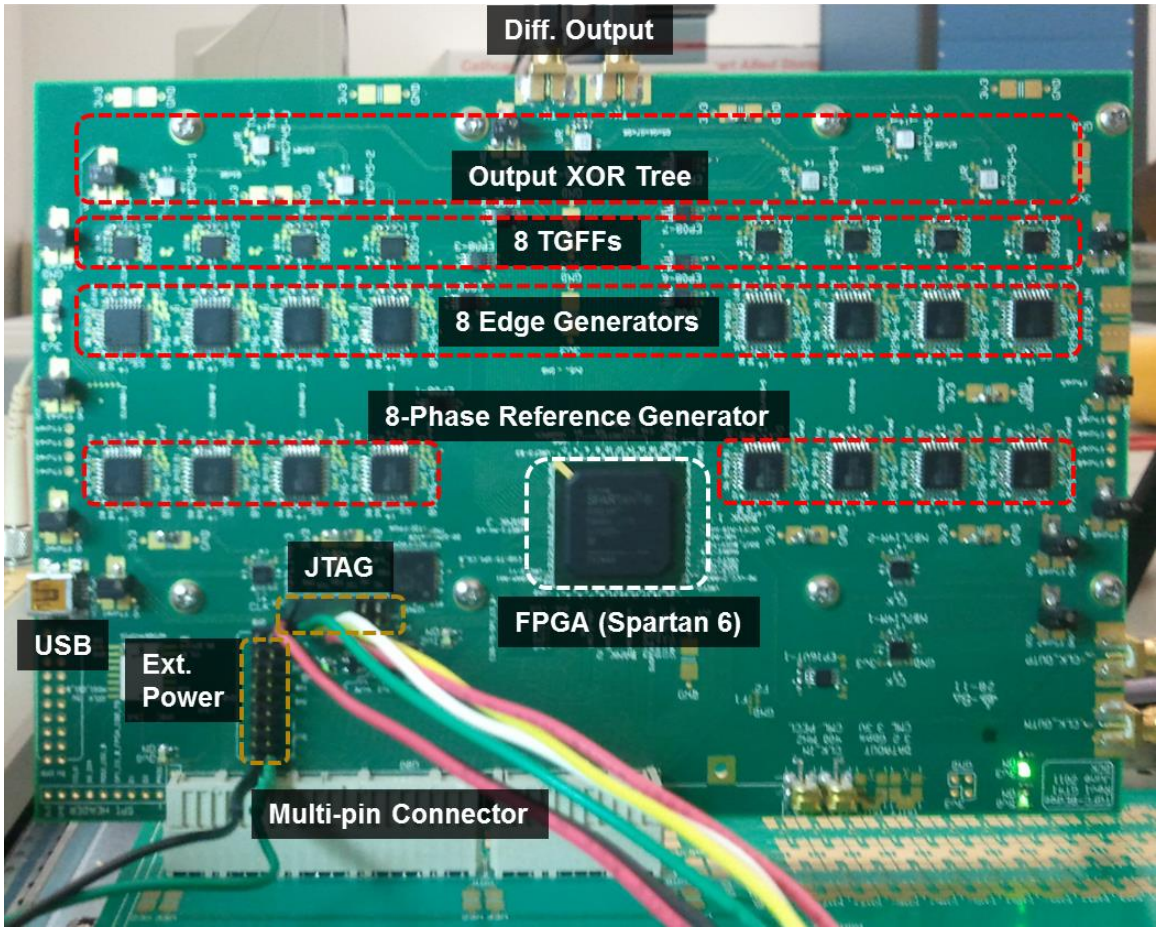


Figure 3.31: Photograph of ATG working on HP83000-660i ATE system.

The ATG algorithm was implemented by Verilog language, the behavioral simulations and post-route verifications were completed under Modelsim 7.0 and Xilinx ISE 13.0 respectively. All the following measurements shown in this section were measured by a Keysights 86100C high-speed sampling oscilloscope with 54752A plug-in testing module (26/50GHz bandwidth), and the reference clock, power supplies and fetching pointers were all provided by HP83000-660i ATE. The data pattern is provided by either ATE or PRBS-31 generator built inside FPGA.

3.5.1 Basic ATG Operation: 3.2Gbps PRBS data transition

The very basic capability of the ATG algorithm is to produce timing signals with any user-defined pattern at fix and full data rate 3.2Gbps. This is illustrated in Figure 3.32, where all the fine-delay edge generators are programmed to be at a fix delay value, also the pattern is programmed to PRBS-31 (Length=2^{31}-1). Notice that the edges line up almost exactly with the oscilloscope grid markers, which is set at 312.5ps intervals.

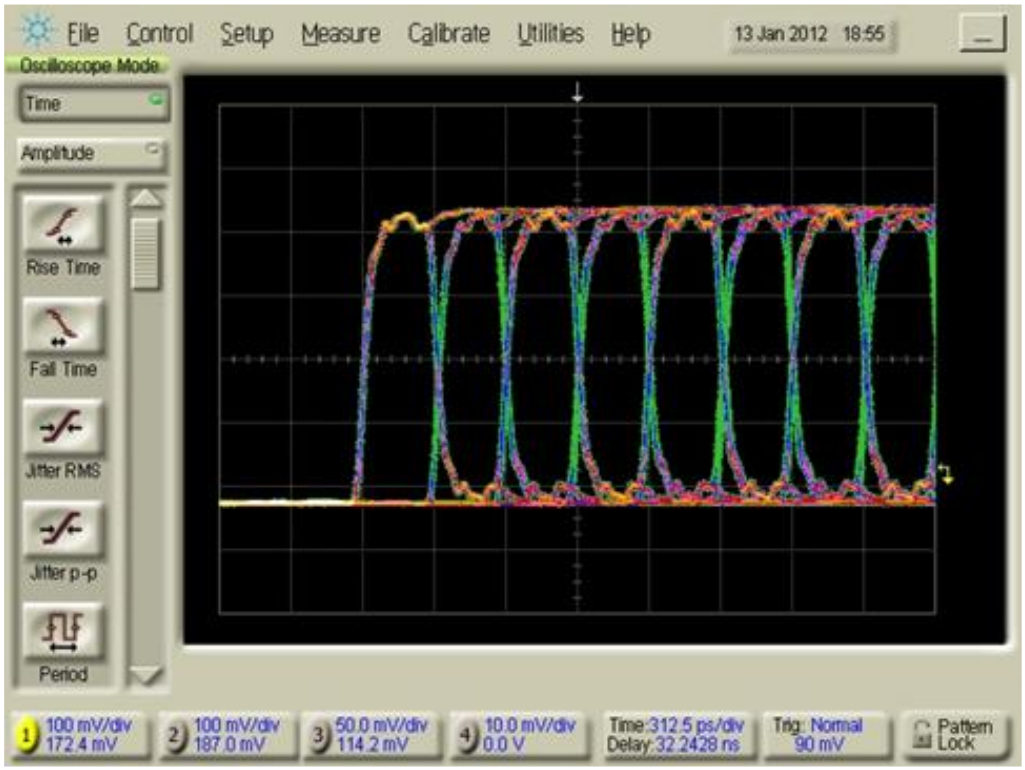


Figure 3.32: 3.2Gbps eye diagram using PRBS-31 pattern. Time base is 312.5ps/div.

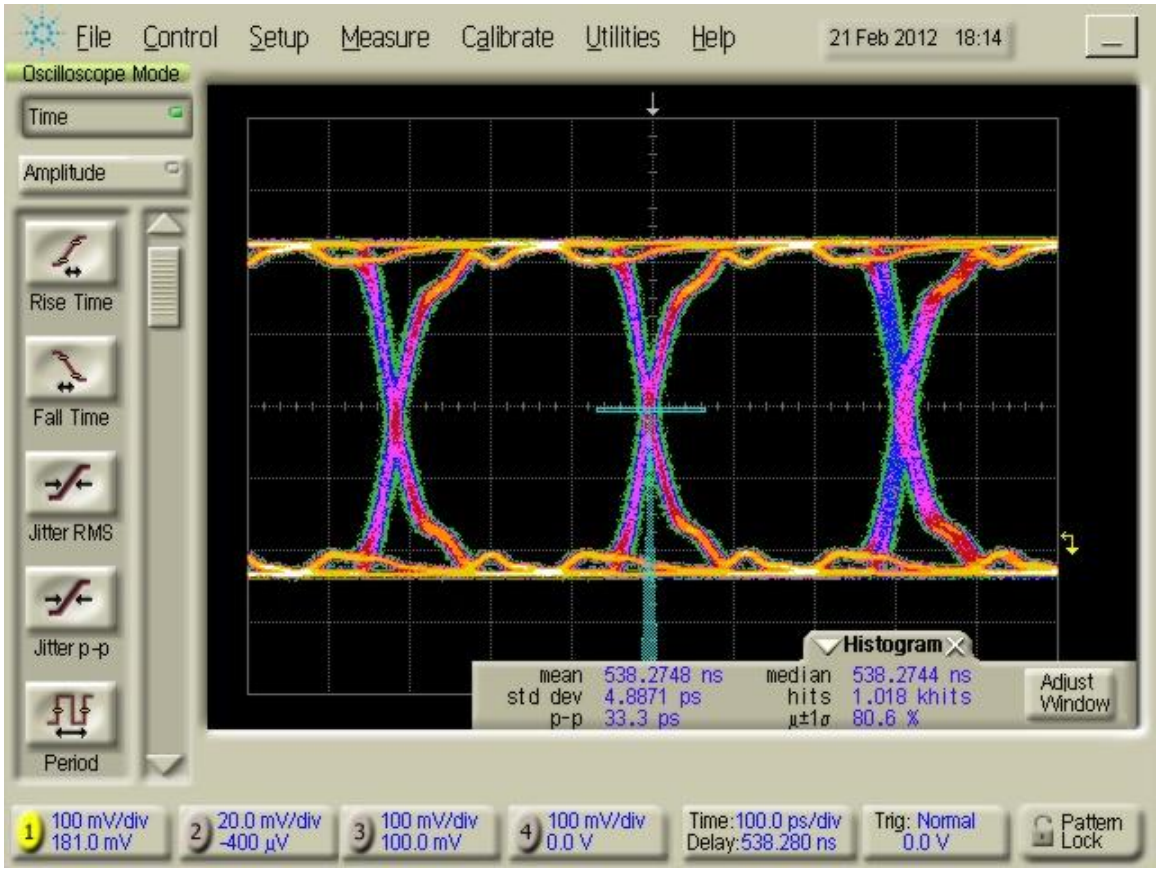


Figure 3.33: ATG eye diagram using 3.2Gbps PRBS-31 pattern. RMS jitter is ~4.8ps and total Jitter is ~33ps (1khits). Time base is 100ps/div.

We later measured the data-dependent jitter (DDJ) to be on the order of 6-12ps, which is very small, especially considering the three-stage XOR gates (total 7 XORs) for serializing in the HS section. Random jitter (RJ) presents in Figure 3.33 is about 4ps RMS and is dominated by the 400MHz master clock jitter from ATE and oscilloscope trigger signal (also from ATE). Therefore we can conclude that very small amount of RJ is added by the ATG. All this is done while the other edges remain precisely spaced at 312ps intervals to produce 3.2Gbps. For this measurement, we used "infinite persistence" display mode and collect the data for at least 1K hits, so the ~30ps peak-to-peak (pp) random jitter (~6σ) is observed as a widening of the signal trace.

3.5.2 Edge phase shift and jitter injection

In Figure 3.34, we demonstrate the ATG control of a single edge (without affecting neighboring edges) by re-programming the fine delay for one of the edge generators each time the test is re-run within a real-time looping pattern. The figure shows one of the edges incrementing in approximately 35ps steps through four values (0, 35, 70, 105ps). In fact the system are able to move the edge in even smaller steps (down to 10ps). However, in this time scale with 15ps pp jitter, it might be very difficult to be observed on scope.

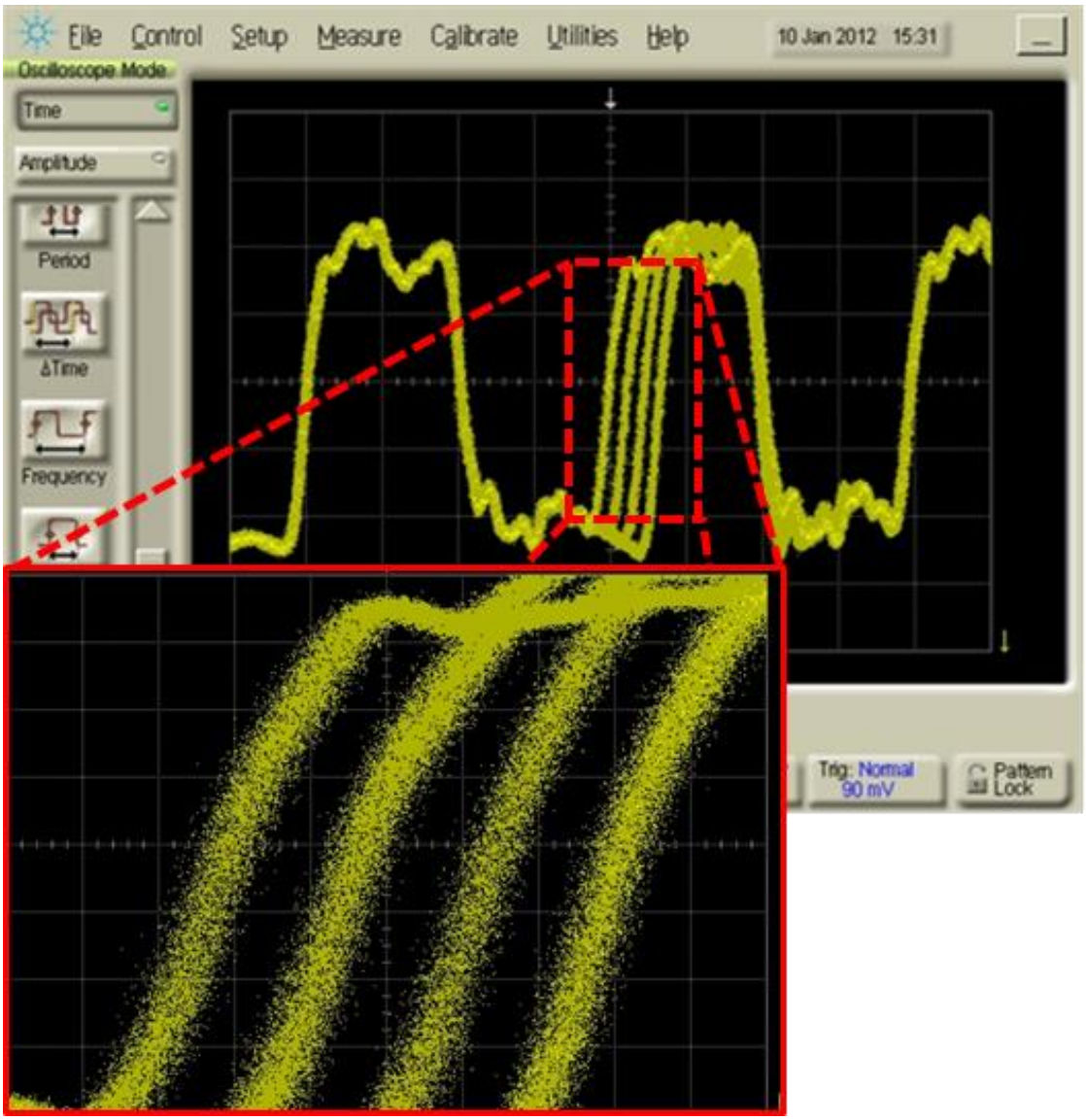


Figure 3.34: ATG demonstration of individual-edge "timing-on-the-fly." The third edge (center of screen) is delayed in ~35ps steps, while others remain unchanged. Time-base is 156ps/div and zoom in part is 20ps/div.

Individual edge-shift is demonstrated in Figure 3.35. In this experiment we program the first, third, and seventh edges of a 3.2Gbps signal with fine-delays of 210ps, 110ps, and 160ps respectively. Since the nominal delays of the other edges are at zero (relative to the 312ps references), we end up with "pulses" of varying widths, namely 100ps, 200ps, 312ps, 152ps. Also, the very last pulse on the screen is generated by edges 1 and 2 in the next group of 8. Notice that these are now reprogrammed to their nominal settings (0 and 312ps), so the very last pulse width is again back to 312ps. The first pulse (~100ps width) is nearing the minimum of ~78ps that can be produced with the ATG algorithm. At 100ps, this period is equivalent to a single-bit at 10Gbps, this short pulse shows the ATG has the potential to run a lot faster than the spec claims with current hardware.

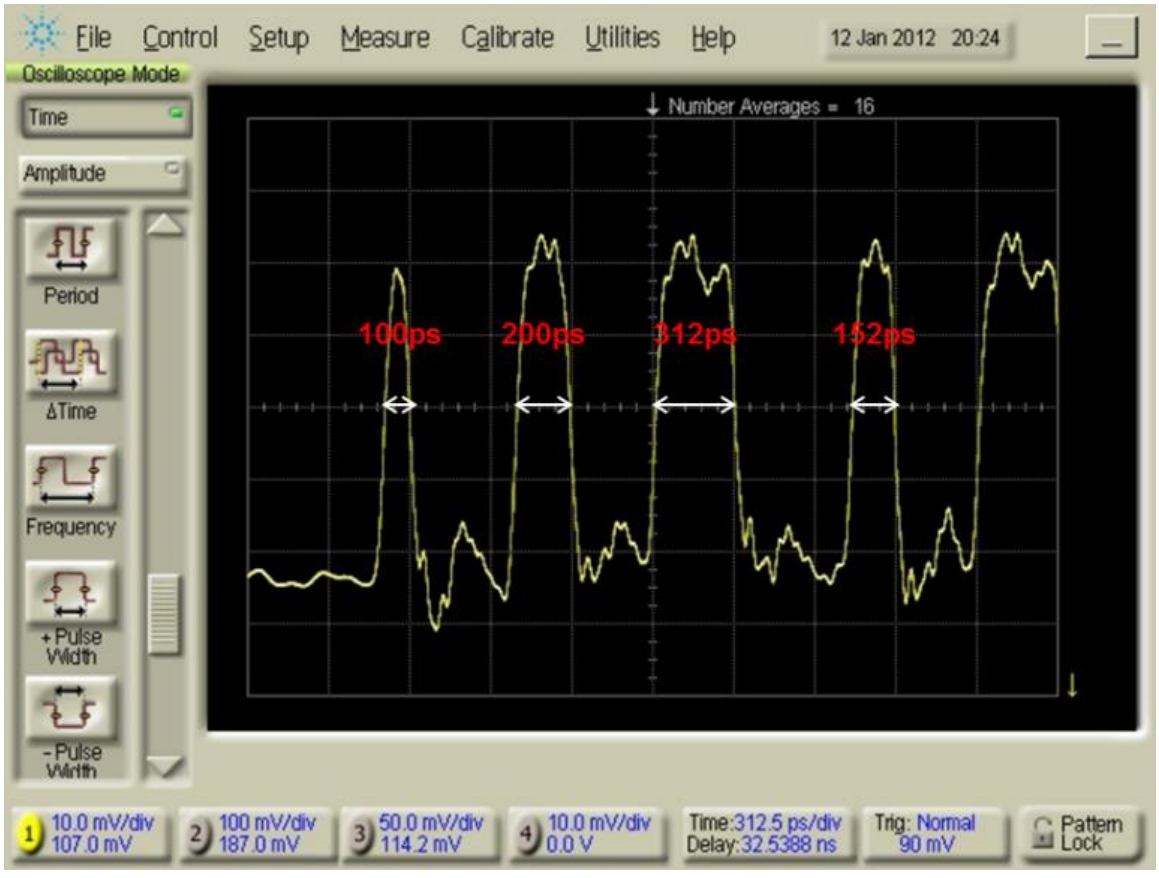


Figure 3.35: Multiple-edge "timing-on-the-fly." The first, third, and seventh edges are delayed by 210ps, 110ps, and 160ps respectively. Time-base is 312ps/div.

Since the ATG is able to shift individual edge, the algorithm is able to modulate jitter injection by programming different multiple delay values at the same data rate. The modulation frequency can be up to 400MHz since the data in edge generator is refreshed every 2.5ns. Figure 3.36 demonstrates the ATG ability to synthesize extra DDJ. Here for a good observation on scope, the timing edges are adjusted to create the effect of about +/- 50ps (100ps peak to peak) of added DJ.

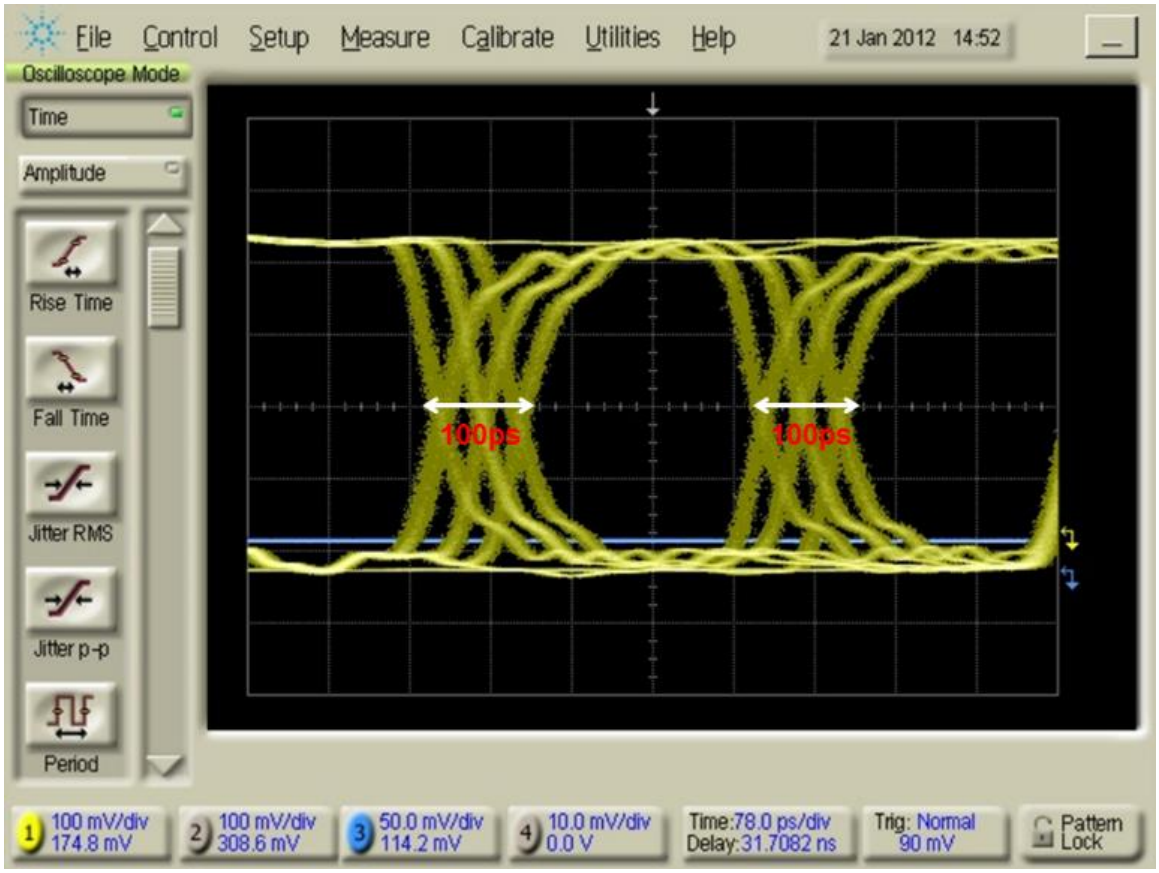


Figure 3.36: DDJ jitter injection. The signal bit-rate is 3.2Gbps. Time-base is 78ps/div.

3.5.3 Low-speed and long-delay demonstration

While previous section shows the individual timing-shift ability that can be performed by ATG algorithm, it doesn't really show a significant advantage over ATE system. A more challenging application is illustrated in Figure 3.37 where the capability of timing-on-the-fly is used to synthesize a 1.0Gbps signal. Notice that 1Gbps is NOT an

even multiple of our reference clock (400MHz). Therefore, each edge must be reprogrammed with a new "fine-delay" value every time the generator is used. In fact, each of the edges shown in the example waveform is coming from different edge generator, and delay value in each edge generator changes every time when it is used by the ATG. As shown in Figure 3.37, the edge E1 is generated from edge generator D7, E2 is generated from D2 of next output cycle and etc. Despite the complexity involved, the resulting waveform has shown a very precise timing and low-jittery 1.0Gbps signal.

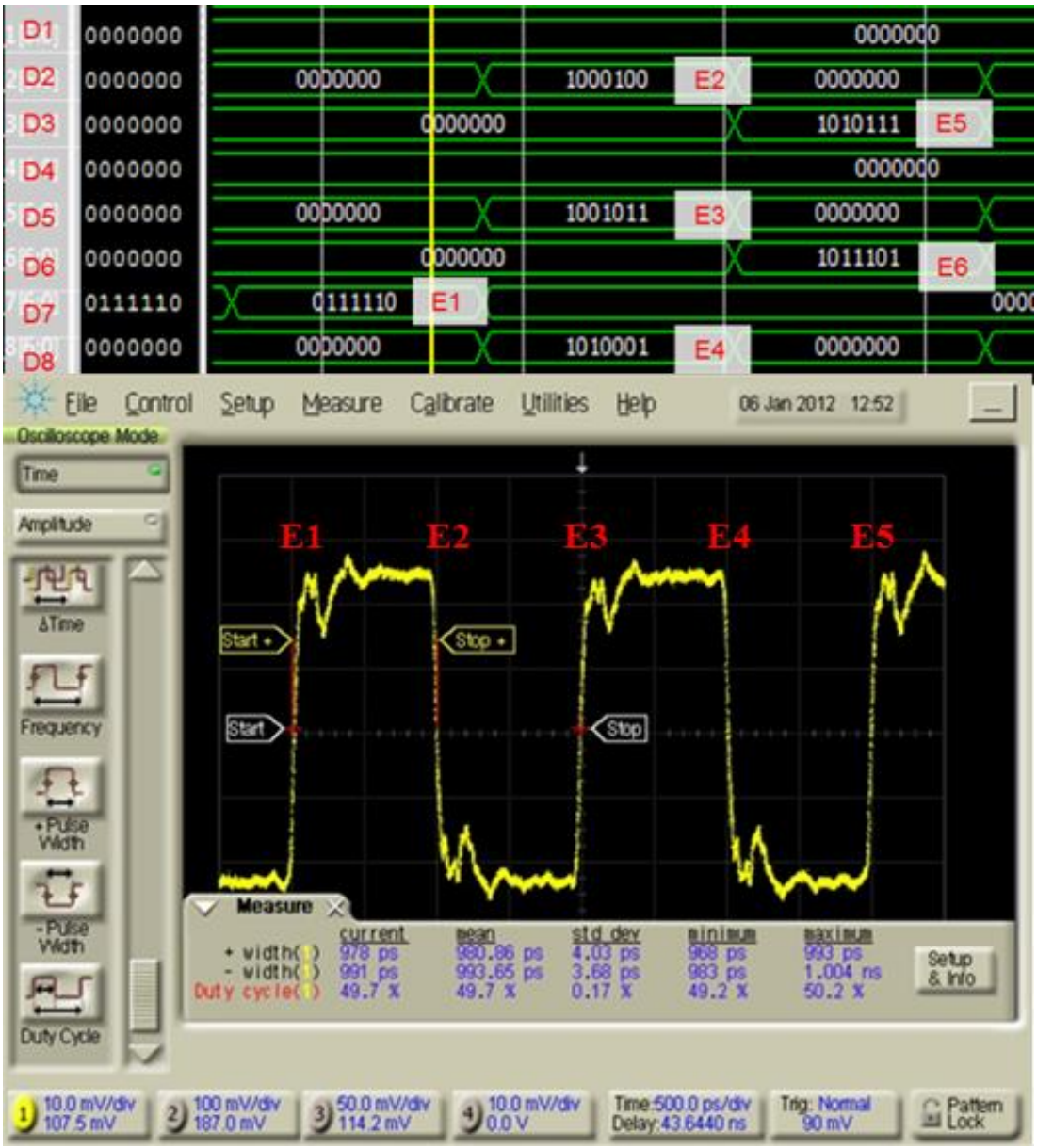


Figure 3.37: Synthesizing 1.0Gbps using "timing-on-the-fly." Time-base is 500ps/div.

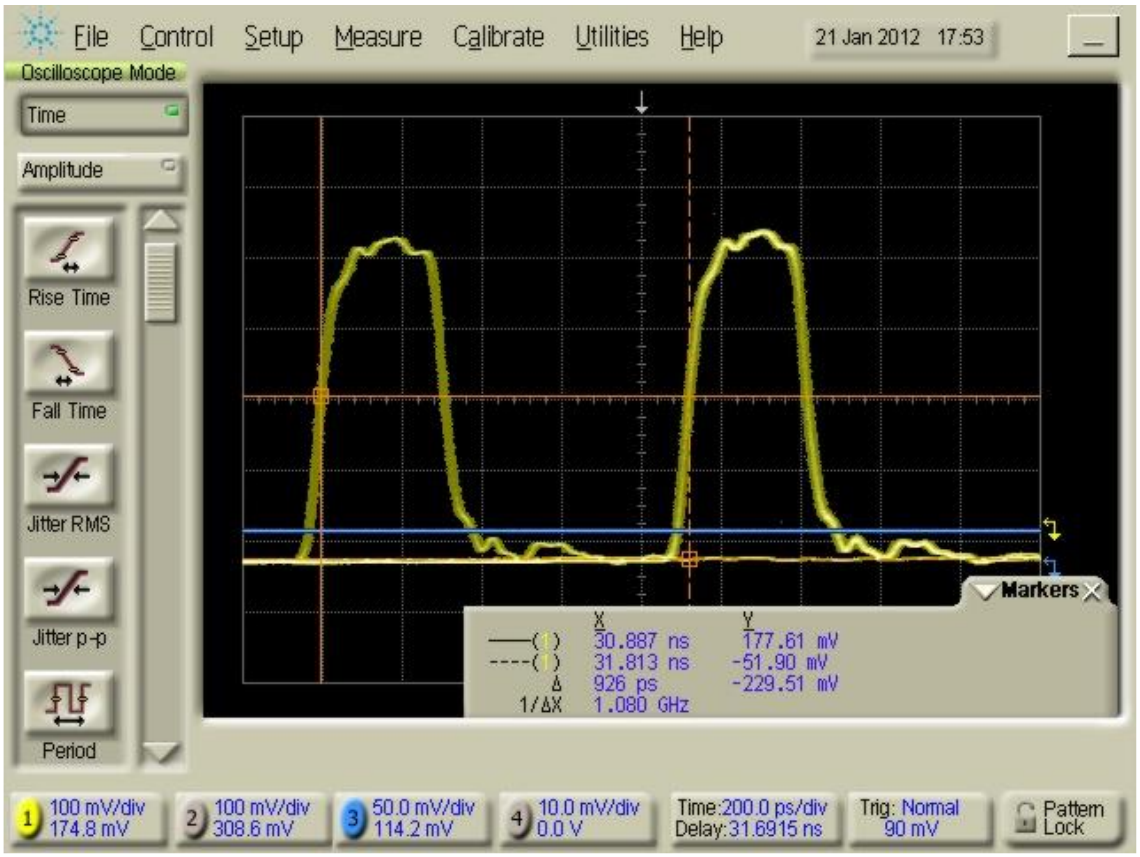


Figure 3.38: Demonstration of the ~1ns dynamic programming range. A 312ps wide pulse is shifted by 937.5ps. Time-base is 200ps/div.

Ideally each of the 8 edge generators must cover at least a range of 312.5ps. However, in order to completely avoid any dead-zones while running at different data rate, and to allow for burst-mode synthesis of higher-speed signals, we make each generator to span an even larger range of 937.5ps. As shown in Figure 3.38, the first edge is generated by E1 with 0 delay and the third edge is also generated from E1 but with 937.5ps delay. The other critical issue of creating long-delay edge is the propagation delay inside the edge generators. Since the edge decoder updates delay value every 2.5ns, the edge generators must generates the edge delay before the next delay value coming, otherwise the delay value is refreshed by newer value and causes incorrect edge. Therefore the cycle and phase control inside the algorithm needs to be calculated accurately. An easy example is shown in Figure 3.39, which we program a maximum fine delay of ~940ps (the worst case of current cycle might be overwritten by next cycle before current edge is generated) that an

edge generator performs to verify this potentially issue. In reference waveform (in yellow words), all the E1 and E2 are programmed to identical 940ps delay (D1 and D2, respectively) but in different cycles. In other words, E1 cycle1 is generated by 1st 400MHz HS cycle and E1 cycle2 is generated by 2nd one. For the target waveform (in red words), we perform the cycle2 in different values, and the delay time is shorter (~200ps) than the reference one. We can observe the E1 and E2 in cycle1 of target waveform is not affected (E1 cycle1 and E2 cycle1 of target waveform is overlapped with reference one) by the different delay values in cycle2. This result shows the precise phase control from the algorithm effectively prevents the possible timing conflicts in ATG hardware.

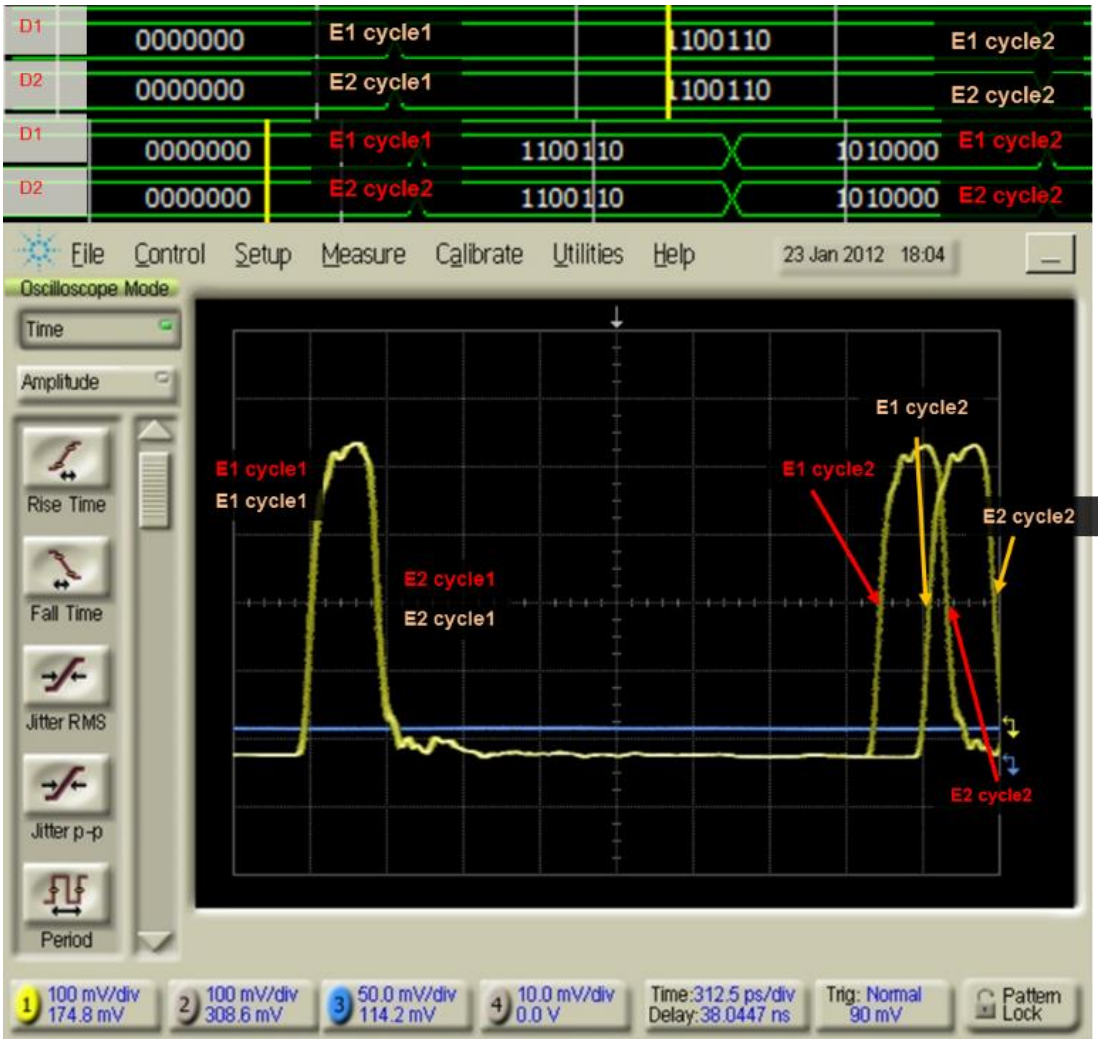


Figure 3.39: Precise cycle and phase control in the algorithm. Showing the algorithm prevents potential timing conflicts from ATG hardware.

3.5.4 Timing-on-the-fly and Burst mode

In Figure 3.40 we show a truly remarkable ability of the ATG to switch from one frequency (period) to another "on-the-fly" without any dead time or glitches. This ATG board is able to fully perform this ability with no constraint and 10ps resolution under the maximum data rate 3.2Gbps. In this example, we program first 6 cycles at 3.0Gbps, followed immediately by 2 cycles at 1.0Gbps, then 2 cycles at 3.0Gbps, and 2 cycles at 1.0Gbps, etc. The data pattern is programmed to clocked pattern (0101…). This capability of freely switch period/frequency on-the-fly at multi-gigahertz rates is not provided by other ATEs today. By help of this ATG, we finally can achieve the ability of the software simulation in hardware testing. Another example we program one 3.0Gbps, two 2.0Gbps, two 1.0Gbps and two 3.2Gbps cycles with PRBS-31 pattern is shown in Figure 3.41.

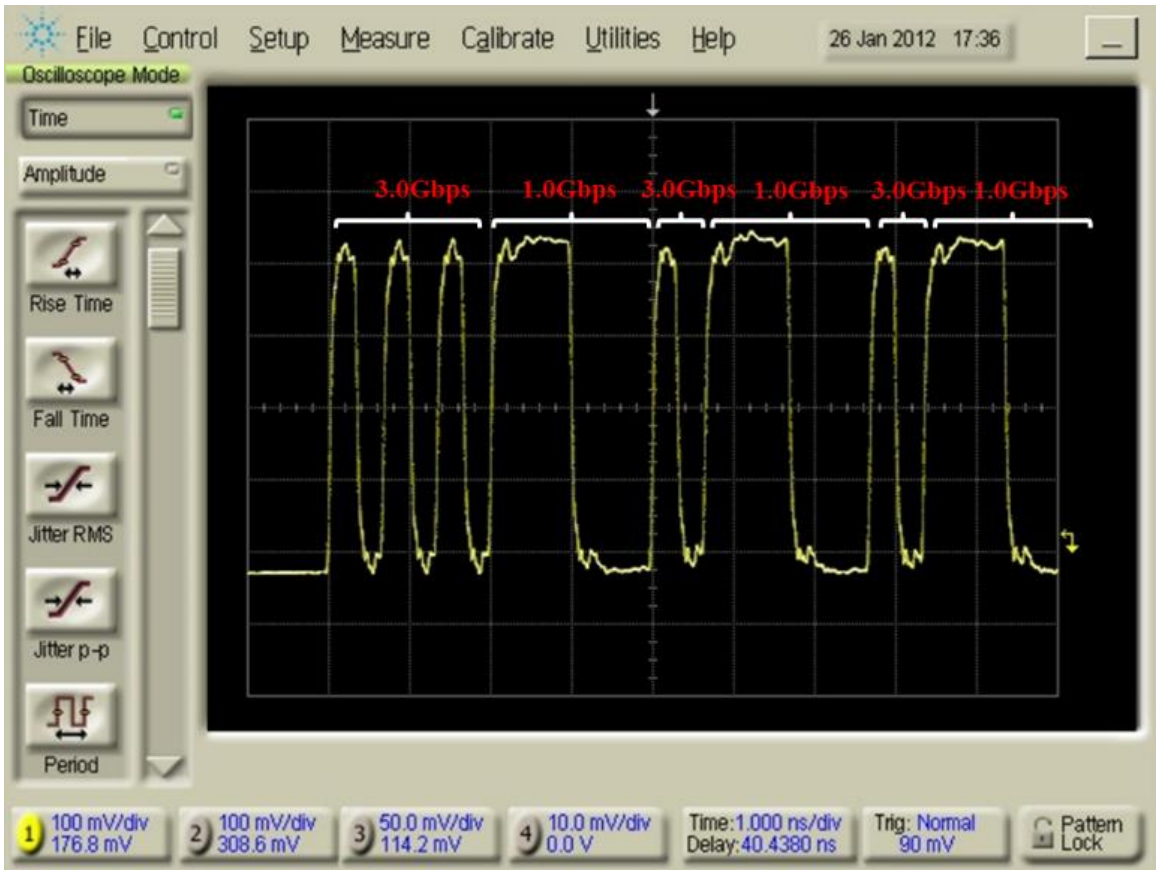


Figure 3.40: A demonstration of "period-on-the-fly" with clocked pattern ("0101…"). Time-base is 1ns/div.

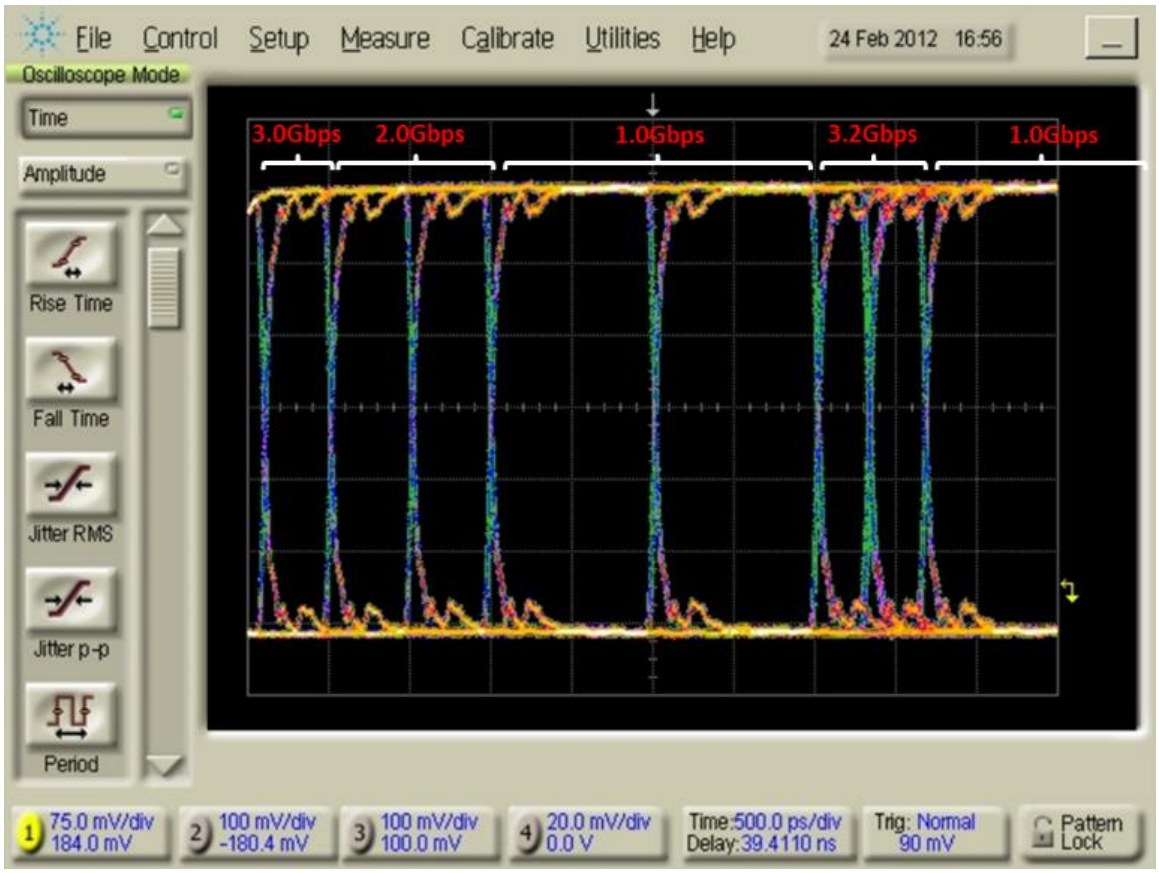


Figure 3.41: A demonstration of "period-on-the-fly" with PRBS-31 pattern. Time-base is 500ps/div.

The other function of the ATG algorithm is to perform the burst mode which allows to exceed typical operation data rate (3.2Gbps). The concept is to use the overlapping area of the edge generators to generate maximum edges in a very short period. As we mentioned before, every 312.5ps overlapping area is covered by three different generators, therefore theoretically we are able to synthesize as small as 78ps (312.5/4) width pulse. If several edge generators are programmed within a short time interval, then bursts can be obtained above 3.2Gbps. Figure 3.42 and Figure 3.43 use this approach to demonstrate the signals running at 5.0Gbps and 6.4Gbps respectively. The jitter measurement at these over-spec data rates still shows a reasonable number compare to typical 3.2Gbps, which means the HS components is able to run a lot faster. However, this ability of running at the data rate above 3.2Gbps is limited. The ATG algorithm actually sacrifices the flexibility of

producing continuous bit stream and no dead-zone edge ability. In other words, the ATG system only produces several data eyes instead of continuous eyes in burst mode. In Figure 3.43, the four edges to create three 6.4Gbps data eyes (156.25ps eye-width) have burned out the usage of four nearby generators so that none of the edge can be generated within 937.5ps (the next available generator) thus a continuous bit stream of high data rate is not achievable. The other issue of running burst mode is the limitation of rise/fall time of the XOR gate (which is specified at 20ps). When ATG is operating the data rate over 10Gbps (100ps period), the signal cannot perform full voltage transition from low to high level. Figure 3.44 demonstrates the ATG ability to synthesize intentional glitches, with pulse-widths less than 100ps. The first two burst pulses are programmed to 78ps (the minimum pulse can be generated by ATG), follow three are programmed to about 100ps and last one is a typical 3.2Gbps cycle to show the reference of full voltage transition. From the figure we can observe that the first five high-speed pulses actually cannot reach Vmax.

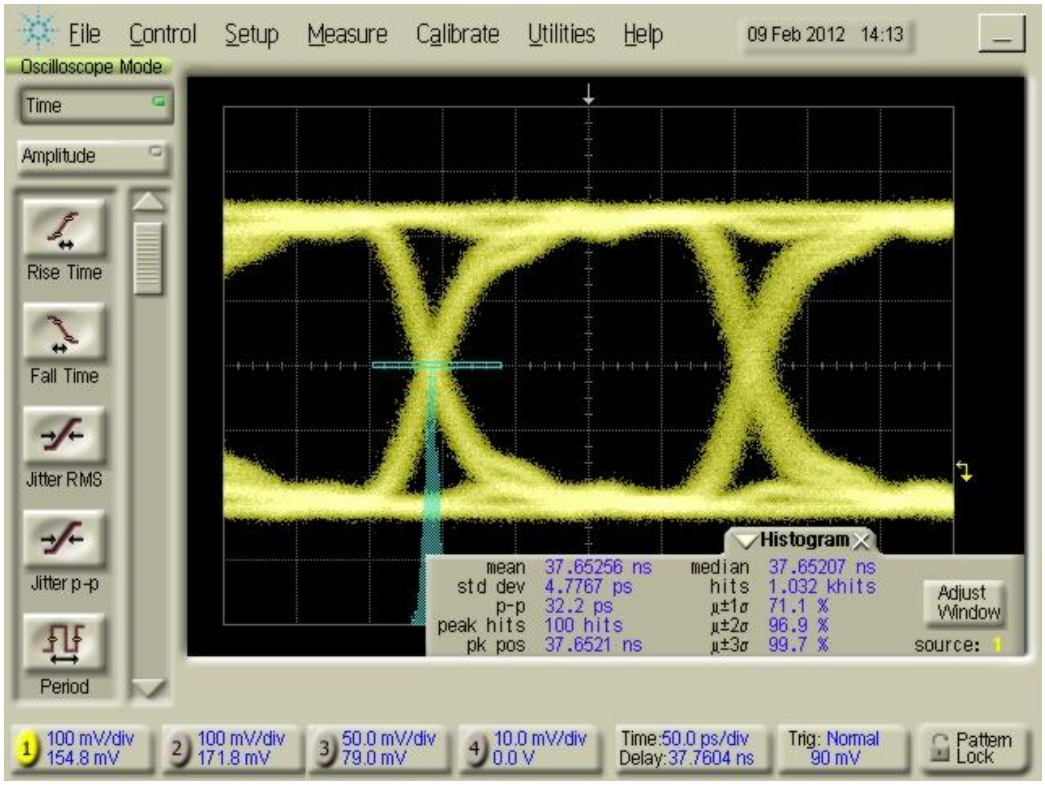


Figure 3.42: Demonstrating synthesis of 5.0Gbps burst data eyes. RMS jitter is about 4.7ps and peak to peak jitter is 32ps. Time-base is 50ps/div.

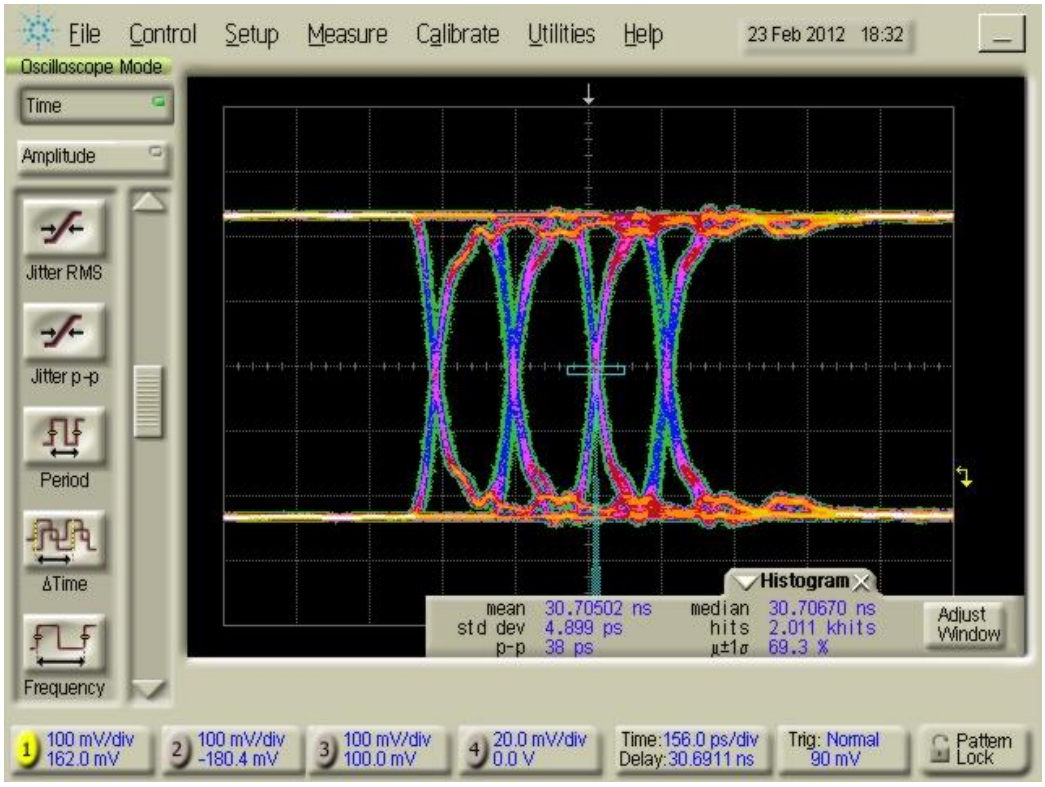


Figure 3.43: Demonstration of 6.4Gbps burst data eyes. Time-base is 156ps/div.

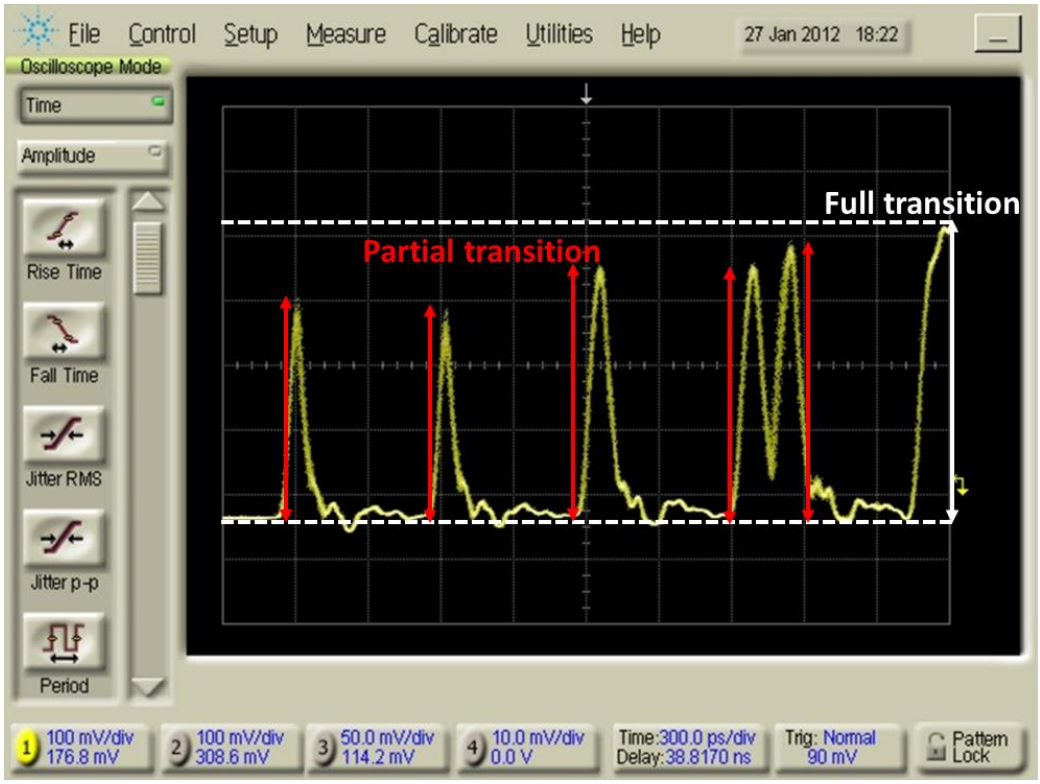


Figure 3.44: Synthesizing glitches with pulse-widths <100ps. Time-base is 300ps/div.

3.5.5 Rising/Falling time improvement and MUX Mode

To solve the rise/fall time limitation, we use an edge-sharpening XOR buffer (Hittite HMC844) to increase the ATG output slew rate and to suppress noise on the logic rails. When driven by the ATG output, this buffer produces a 20-80% rise time of about 12ps, as measured with our instrumentation. This is illustrated in Figure 3.45 and is very close to the device specification (10-11ps). Therefore, sharper-edge components is very helpful if we want to push this ATG to higher data rate in the future.

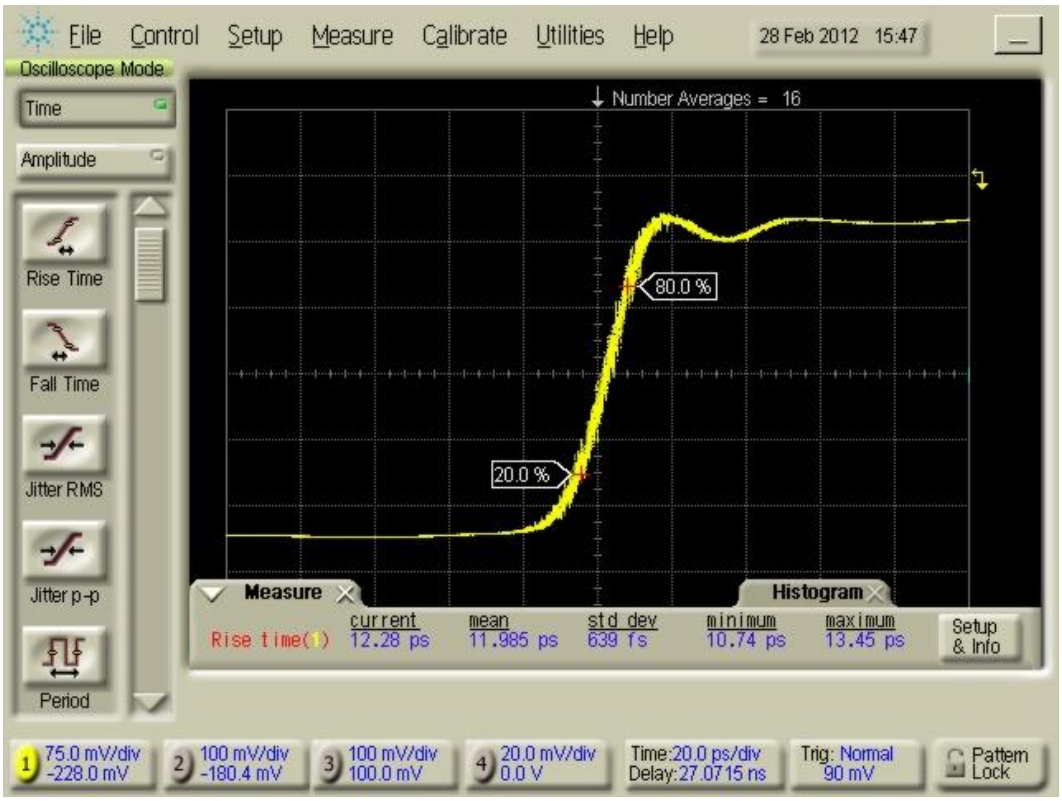


Figure 3.45: Buffered output edge. T_{RISE} = 12ps (20-80%). Time-base is 20ps/div.

Figure 3.46 shows the sharper rise/fall edge at 1.0Gbps from buffered ATG output. The target F_{MAX} of 3.2Gbps for the buffered ATG is shown in Figure 3.47. Here two consecutive data eyes are visible and the total jitter for one edge is about 29.5ps at BER=10^{-3}(1khits). At this BER, the eyes openings are about 0.91UI wide. Given an estimated DDJ = 10ps, then RJ = 1/6(29.5-10) = 3.25ps. Therefore, at BER = 10^{-12} we expect TJ = 56ps, and the eye opening is about (312-56)/312 = 0.80UI. This includes the ATE clock

reference jitter, the oscilloscope trigger jitter, and that of the added buffer, as well as the ATG jitter itself.

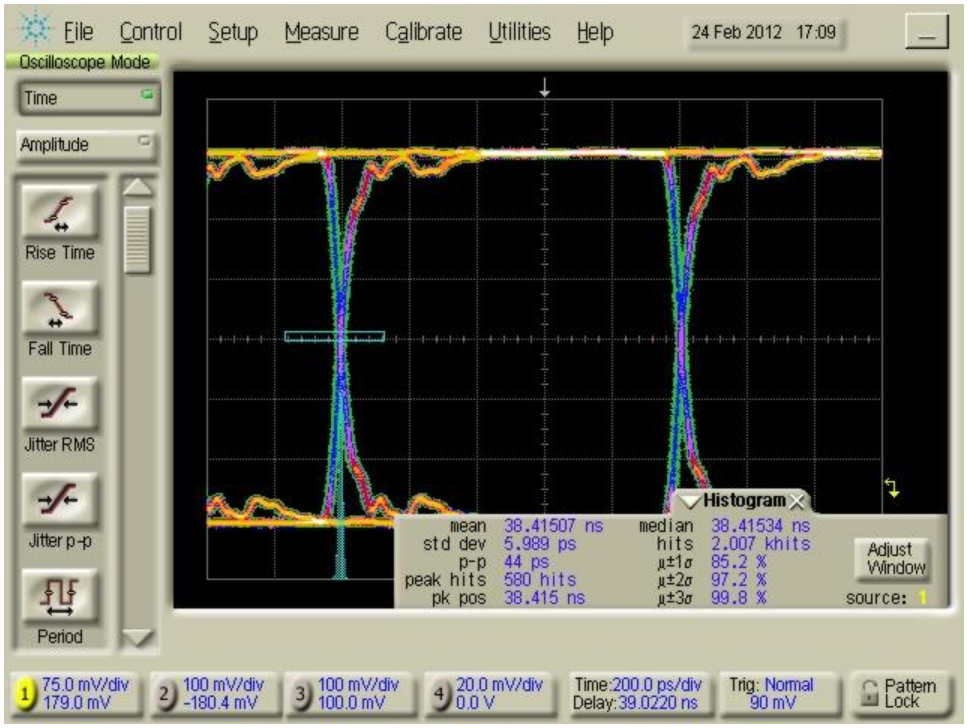


Figure 3.46: Buffered output eyes at 1.0Gbps. Time-base is 200ps/div.

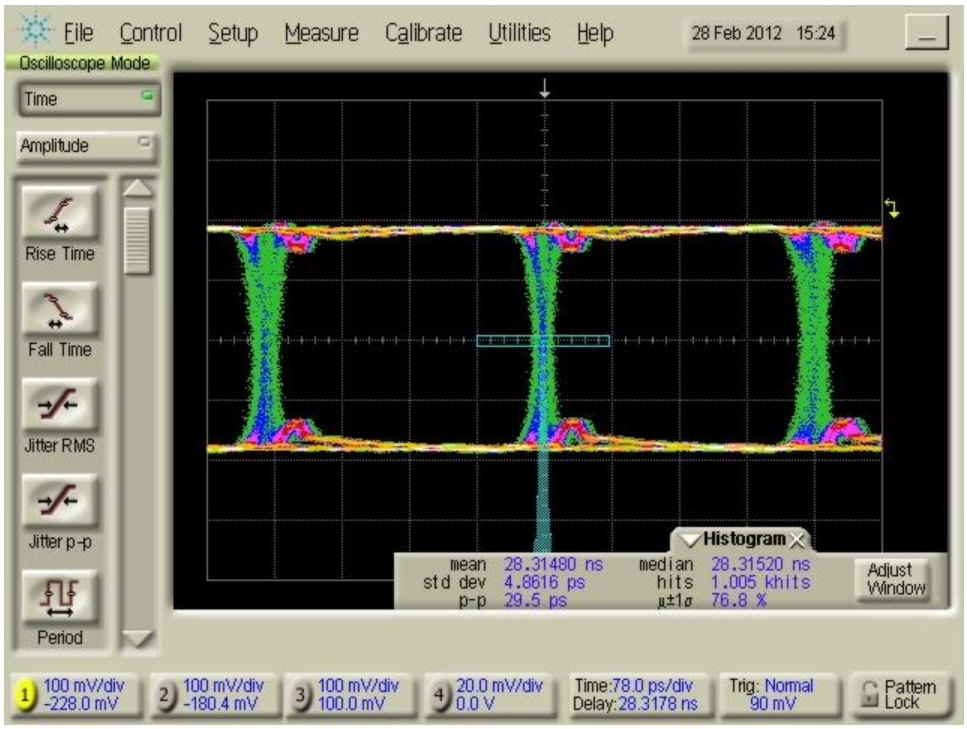


Figure 3.47: Buffered output eyes at 3.2Gbps. TJ = 29ps.Time-base is 78ps/div.

In this demonstration, the ATG shows the ability to combine two single-ended ATG signals (each running at 3.2Gbps) using the HMC844 (up to 20Gbps) as another level of exclusive-OR. Since the component supports differential inputs/output, the method is to operate the MUX in single-ended mode. Each single-ended ATG signal goes into one of the XOR inputs and the other end of XOR inputs is 50 ohm terminated to ground, the schematic of this setup is shown in Figure 3.48.

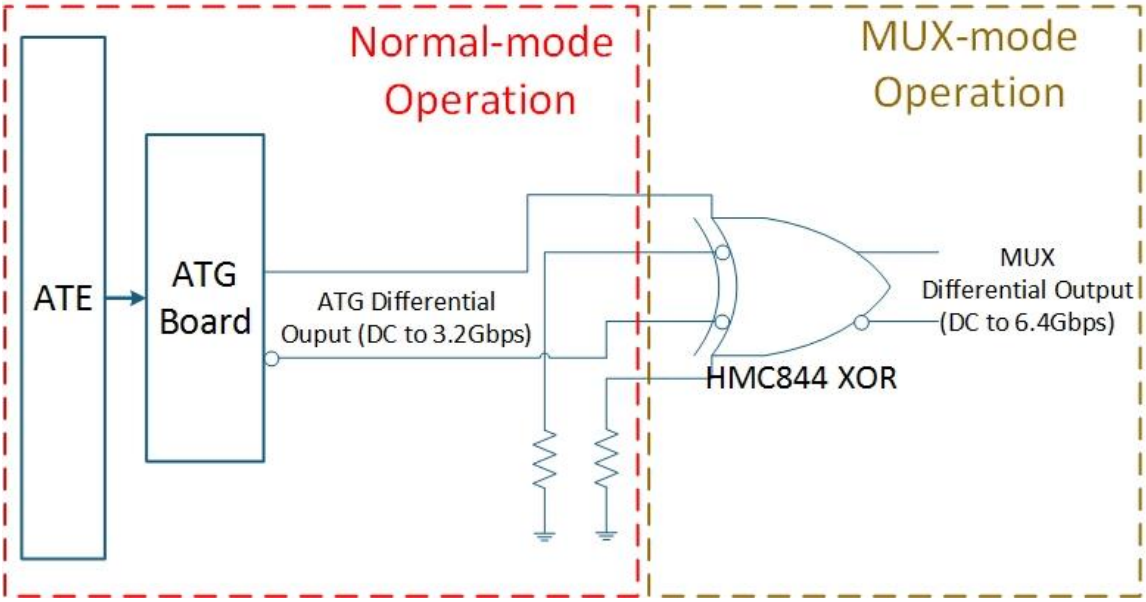


Figure 3.48: The schematic of implementing MUX mode on ATG board.

This approach essentially expands the ATG to 16-to-1 multiplexing (originally 8-to-1 multiplexing), and extends the typical F_{MAX} to 6.4Gbps (sustained, not burst), as illustrated in the Figure 3.49. We can observe four complete and continuous eyes are visible in the figure and jitter characteristics are comparable to previous examples. At last, we try to combine both burst mode and MUX mode to push the maximum data rate of this ATG. In Figure 3.50, a burst of 3 isolated data eyes of 10Gbps are shown (no other edge is surrounded since we need to sacrifice some capabilities of ATG in burst mode). Here the long (~937.5ps) dynamic range of the edge generators is exploited in order to force four edges to occur within 312.5ps. By the mixture of MUX and burst mode, the ATG is able to synthesis a short burst of data up to 10Gbps with sharp rise/fall time.

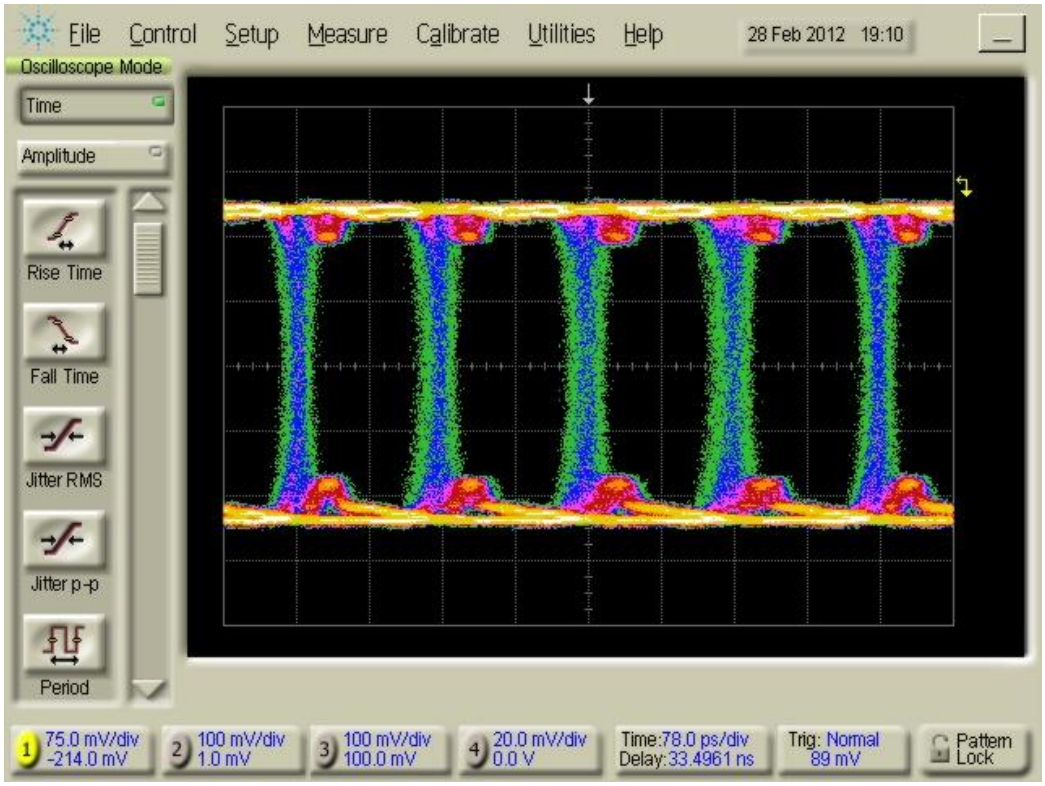


Figure 3.49: Multiplexed/buffered output eyes at 6.4Gbps. Time-base is 78ps/div.

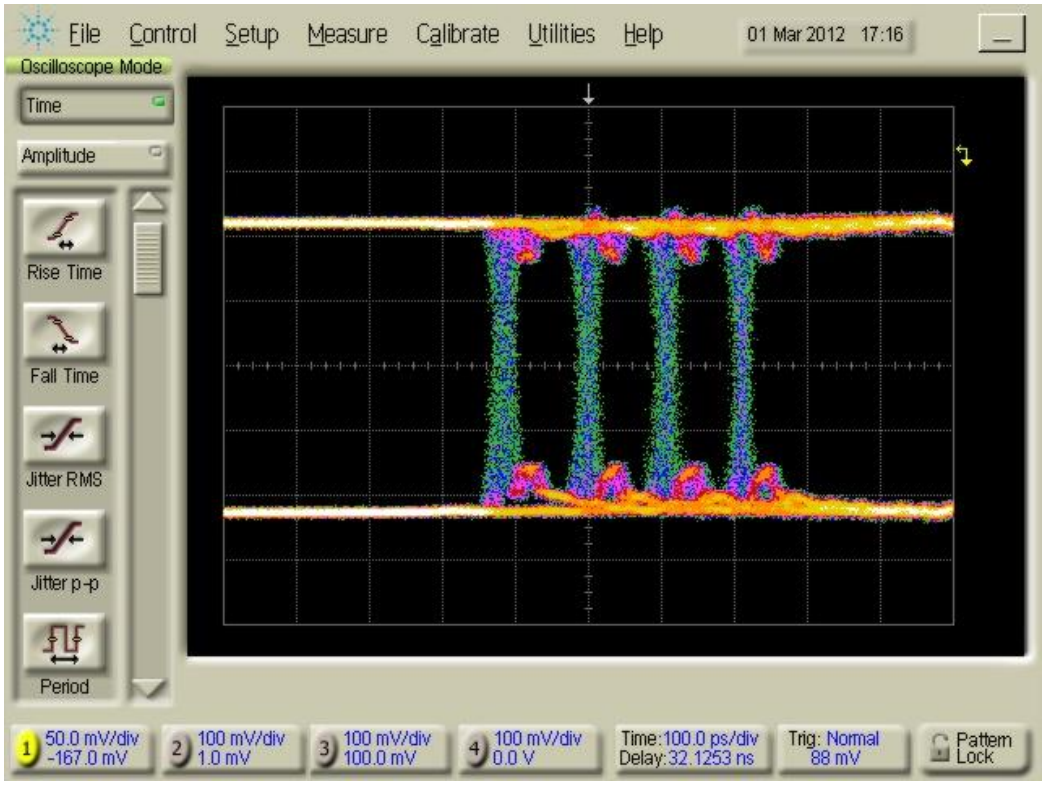


Figure 3.50: A burst of 3 bits at 10Gbps (buffered output). Time-base is 100ps/div.

3.6 Summary

This ATG provides the capability of switching period/frequency “on-the-fly” at multi-gigahertz rates which is not provided by current ATEs. This FPGA-based application has raised the flexibility of DUT test to a higher level. However, the ATG is still connected with an ATE system and requires control lines and power supplies from ATE, therefore currently this ATG can only be considered as an extension module of ATE and cannot operate individually to replace the entire ATE system. Furthermore, although the system can fully operate at 3.2Gbps and go up to 6.4Gbps or higher under some constraints, the 6.4Gbps maximum speed just barely catches up the high performance devices nowadays and might be out-of-date soon. Due to those issues, a more adaptive control host is needed to be implemented to allow users to define their testing data pattern in a more efficient way without the help of ATE. Furthermore, there is still room to optimize the algorithm of HDL code to achieve higher throughput, and a more advanced FPGA chip with larger capability, powerful functions and faster speed will definitely improve the bottleneck we meet today. In short, this ATG is a very cutting-edge design and still we haven’t dug out the whole potential of it yet. The high flexibility of FPGA on this ATG board ensures more possible works can be done in the future to keep hardware testing ability on the same path of most advance digital devices.

CHAPTER 4

FPGA-BASED TESTING PLATFORM

4.1 Introduction

High-performance digital testing for hardware evaluation has become a big challenge. For the past 30 years, ATE with multi-channel outputs has provided a stable solution for various digital testing [24] [29] [30]. However, as the data rate of commercial digital devices go up to several Gbps [31], the complexity of generating high-speed testing pattern has increased exponentially. Also the jitter performance, voltage swing and rise/fall time performance are seriously distorted at this data rate. Although most advanced ATEs are able to reach the speed easily, the high-cost, high power consumption and the testability still present difficulties toward testing these high-speed components today. Therefore, we want to design a new testing architecture with economical cost to replace the conventional ATEs, and also extend this design to the new high-speed digital systems in coming future.

In this chapter we will present a FPGA-based, high-performance testing platform for general wide-speed digital test. This FPGA-based platform is built by a new 28nm-process Xilinx Kintex-7 FPGA enabled with four extended slots on the board which provide hundreds of I/O channels for testing purpose. This design can also be expanded by adding more FPGAs on the current board. Furthermore the firmware inside FPGA had been well-developed to perform transmission/reception of wide-band signals from DC to 10Gbps with the high-speed SERDES. This re-configurable firmware not only designed for high-speed signal transmission/reception, but also provides auxiliary interfaces for communication. Plug-in modules are designed to extend the function of this testing platform by using four extended slots. These plug-in boards will be detailed later in Chapter 5. In this chapter we will focus on architecture of this platform and show an up to 16Gbps PRBS-31 signal from FPGA main board in the testing result.

4.2 FPGA-based Testing Platform Design

4.2.1 The FPGA and GTX Transceivers

Our approach is based on utilizing the FPGA as the testing core for this new platform. The development of semiconductor has made high-performance FPGAs become the leading technology of algorithm/concept verification in semiconductor industry. Today FPGAs are available that utilize 28nm CMOS technology and support I/O rates of ~30Gbps (using dedicated serial I/O pins). Even "standard" I/O channels on these FPGA are able to run up to ~1Gbps. Furthermore, the cost of FPGAs is very low as comparing with customized and sophisticated machines such as ATEs. On the other hand, FPGAs have always had the significant advantage of re-programmability, so that design changes and improvements can be realized without incurring significant re-construction costs. Therefore our main focus is to leverage these features of state-of-the-art FPGAs in order to create a low-cost, but high-performance test system.

4.2.1.1 FPGA overview

The selected FPGA component from Kintex-7 family [46] was introduced by Xilinx. This 28nm FPGA family is representative of leading edge FPGA technology which provides most advanced features in FPGA market including up to 400 moderate-speed I/Os and 32 high-speed GTX SERDES transceivers (>10Gbps). This Kintex-7 FPGA contains some most powerful hard-IP system-level blocks, including 477K high-speed 6-input look-up tables (LUTs) for configurable logics, 18Kb/36Kb individual RAM blocks (supports up to 34Mb), high-performance SelectIO^{TM} technology with support for DDR3 (1866Mb/s), advanced DSP slices (25 x 18 multipliers, 48-bits accumulators and pre-adders) for high-performance filtering and optimized symmetric coefficient filtering, powerful clock management tiles (CMT), combining phase-locked loop (PLL) and mixed-mode clock manager (MMCM) blocks for high-precision and low-jitter reference clock. This features are powered by the very low core voltage of 1.0V, thus consuming relatively low power

and dissipating low heat. The most basic configurable unit inside Kintex-7 FPGAs is LUT. The LUTs can be configured as either one 6-input LUT (64-bit ROMs) with one output, or as two 5-input LUTs (32-bit ROMs) with separate outputs but common addresses or logic inputs. Each LUT output can optionally be registered in a flip-flop. Four such LUTs and their eight flip-flops as well as multiplexers and arithmetic carry logic form a slice, and two slices form a configurable logic block (CLB). Four of the eight flip-flops per slice (one per LUT) can optionally be configured as latches.

The moderate-speed I/Os of the Kintex-7 FPGA series supports most single-ended and differential signal I/O standards. This allows designers to implement low-speed controller and digital control bus to control the devices on main board or plug-in modules. Also the low-speed testing (DC to 1Gbps) can be implemented by using these I/O ports. The supporting standards [40] [47] of these I/Os had been well-discussed in Chapter 3 (Table 3.2) and will not be repeated here. The mostly used I/O standard in this platform to implement control signals is LVCMOS25. The number of I/O pins varies depending on devices and package size. Each I/O is configurable and can comply with a large number of I/O standards. With the exception of the supply pins and a few dedicated configuration pins, all other package pins have the same I/O capabilities, constrained only by certain banking rules. These I/Os in 7 series FPGAs are classed as high range (HR) or high performance (HP). The HR I/Os offer the widest range of voltage support, from 1.2V to 3.3V. The HP I/Os are optimized for highest performance operation, from 1.2V to 1.8V.

The GTX transceivers [48] are the high-speed I/Os (up to 10Gbps) which are built by high-performance ring/LC PLLs, user-programmable drivers/receivers, and SERDES technology. The GTX data rate (500Mbps to 10Gbps) of Kintex-7 FPGA matches to the target performance we set for this design, which is to realize 10Gbps data transition on a low-cost platform. Despite to speed consideration, the more important criteria of selecting an appropriate FPGA is the I/O compatibility. The GTX transceivers support most high-speed I/O standards in the market. Table 4.1 has summarized the high-speed I/O standards

and the speed rate supported by the GTX transceivers in Kintex-7 family. As we mentioned before, the core concept of this testing platform is to use those high-speed GTX I/Os to transmit/receive Gbps signal, therefore a more detail discussion of GTX transceivers will be illustrated in the next section.

Table 4.1: Kintex-7 GTX Supported I/O standards

Standard	Line Rate
Fiber (Single-Rate and Multi-Rate)	4.25/2.125/1.0625Gbps
XAUI, RXAUI	3.125Gbps, 6.25Gbps
Gigabit-Ethernet	1.25Gbps
Aurora (Single Rate and Multi-Rate)	6.25/5/3.125/2.5/1.25Gbps
Serial RapidIO (Single Rate and Multi-Rate)	3.125/2.5/1.25Gbps
SATA 3.0/2.0/1.0	6.0/3.0/1.5Gbps
PCI Express 3.0/2.0/1.0	8.0/5.0/2.5Gbps
CPRI (Multi-Rate)	3.072/2.4576/1.2288/0.6144Gbps
OBSAI (Multi-Rate)	6.144/3.072/1.536/0.768Gbps
3G-SDI, HD-SDI (Multi-Rate)	2.97/1.485Gbps
Interlaken	10.3125/6.25/4.25/3.125Gbps
SFI-5	3.125Gbps
OC12, OC48, OC192	0.62208Gbps, 2.4883Gbps, 9.953Gbps
OUT-1	2.666057Gbps
CEI 6.25	6.25Gbps
10GBASE-R	10.3125Gbps
SFP+ (SFF-8431, SFI)	11.1/10.5187/10.3125/9.9532/9.8304Gbps

4.2.1.2 GTX Transceivers

The GTX transceiver is essentially a high-speed SERDES developed by Xilinx for Kintex-7 family FPGAs. It is a highly power-efficient configurable module that can produce line rate up to 10Gbps. The transmitters support programmable pre/post-emphasis and voltage-swing control, and the receivers provide equalization programming to optimize the received signal integrity. Also, these transceivers have built-in supports of PRBS generator/checker, 8b/10b encoding/decoding protocol, comma alignment, channel bonding, advanced equalizer (DFE and LPM), clock correction and characteristic controls for PCIE interface. The GTX transceiver of the Kintex-7 FPGAs includes the following new or enhanced features compare to the previous generation:

1. 2-byte and 4-byte internal data path to support different line rate requirements, the 4-byte data path is recommended as the line rate goes above 6.4Gbps.
2. Provides both quad-based LC tank PLL for best jitter performance at high line rate and channel-based ring oscillator PLL at low line rate.
3. Power-efficient, adaptive linear equalizer mode called the low-power mode (LPM) and a high-performance, adaptive decision feedback equalization (DFE) mode to compensate for high frequency losses in the channel while providing maximum flexibility.
4. RX margin analysis feature to provide non-destructive, 2-D post-equalization eye scan.

Xilinx has designed the GTX transceivers into quad transceiver columns and place them strategically close to the other transceiver blocks in order to minimize the layout size and power consumption. Figure 4.1 shows an example block diagram of a quad GTX transceivers column [48] in a Kintex-7 FPGA (XC7K325T, which is used in this design) with total 16 GTX channels. The Configuration block provides the access to the reference clock and configurable ports on GTX transceivers and the integrated block combines these signals with PCIE configuration ports. The I/O column are used to load user-defined

configuration from either internal FPGA-generated or external inputs, and mixed mode clock manager (MMCM) column are used to take reference clock form clock input and manage the synchronization and routing of clocks and clocking parameters. These blocks allow the users to have fully communication with GTX transceivers and define the desire settings to GTX. Finally the GTX column routes these configurable settings and reference clock to appropriate GTX channel and performs arbitrary GTX operation.

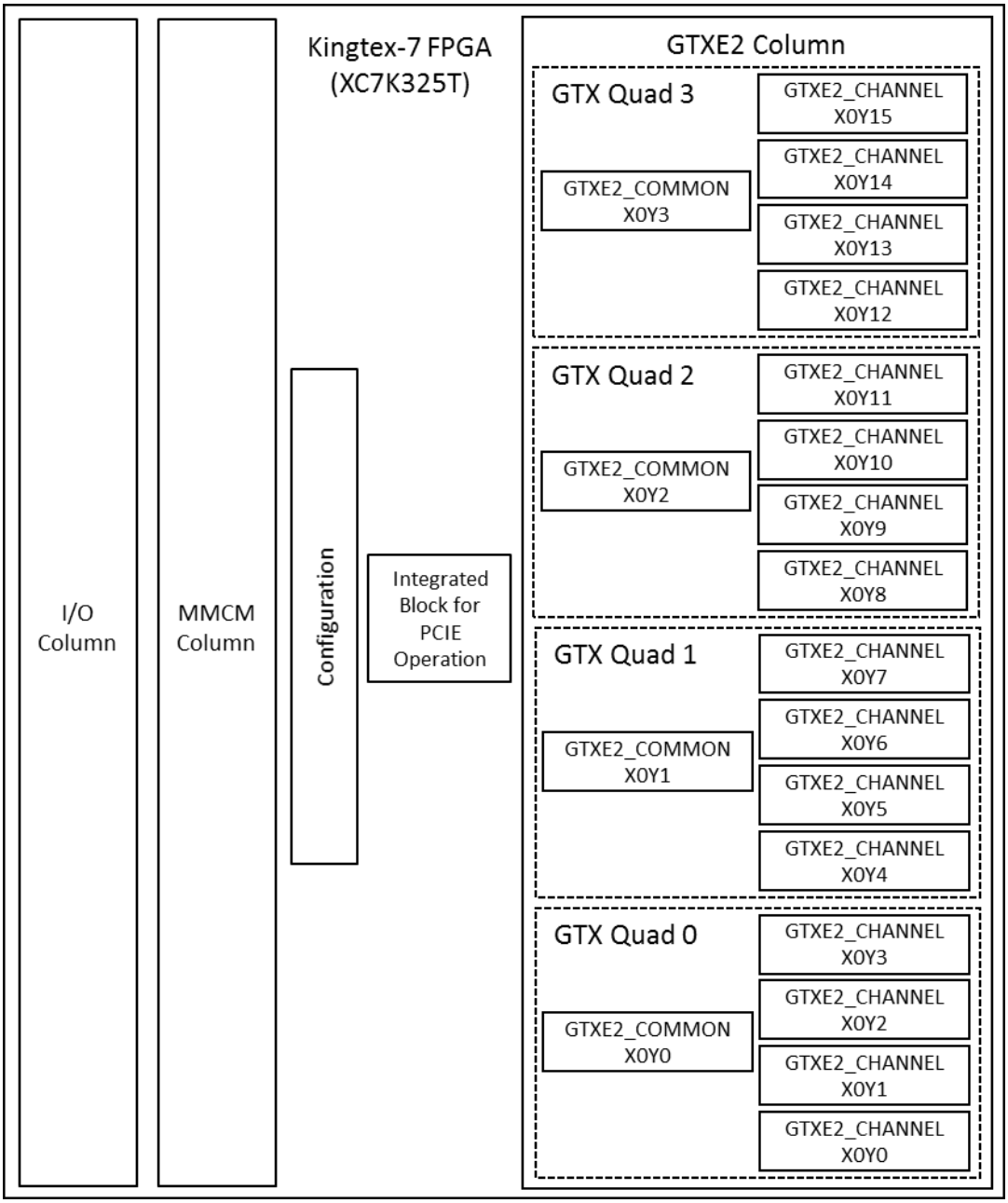


Figure 4.1: GTX Transceiver quad column in a Kintex-7 FPGA (XC7K325T).

GTX transceivers are built into GTX quads, and each quad containing four GTX transceivers as shown in Figure 4.2. Each GTX transceiver has a pair of differential transmitting and receiving ports connected directly to the associated pins on FPGA package. The REFCLK Distribution block receives clock signal from global buffer and distributes this clock signal to either channel PLL (CPLL) or quad PLL (QPLL). Each GTX quad has a LC tank QPLL which provides very low-jitter and high-speed serial clock to all four GTX channels. Despite using QPLL, each GTX channel has a built-in ring CPLL to provide high-speed serial clock for the specific channel usage. Each transceiver channel can be configured to use either one of these two PLL for desired applications.

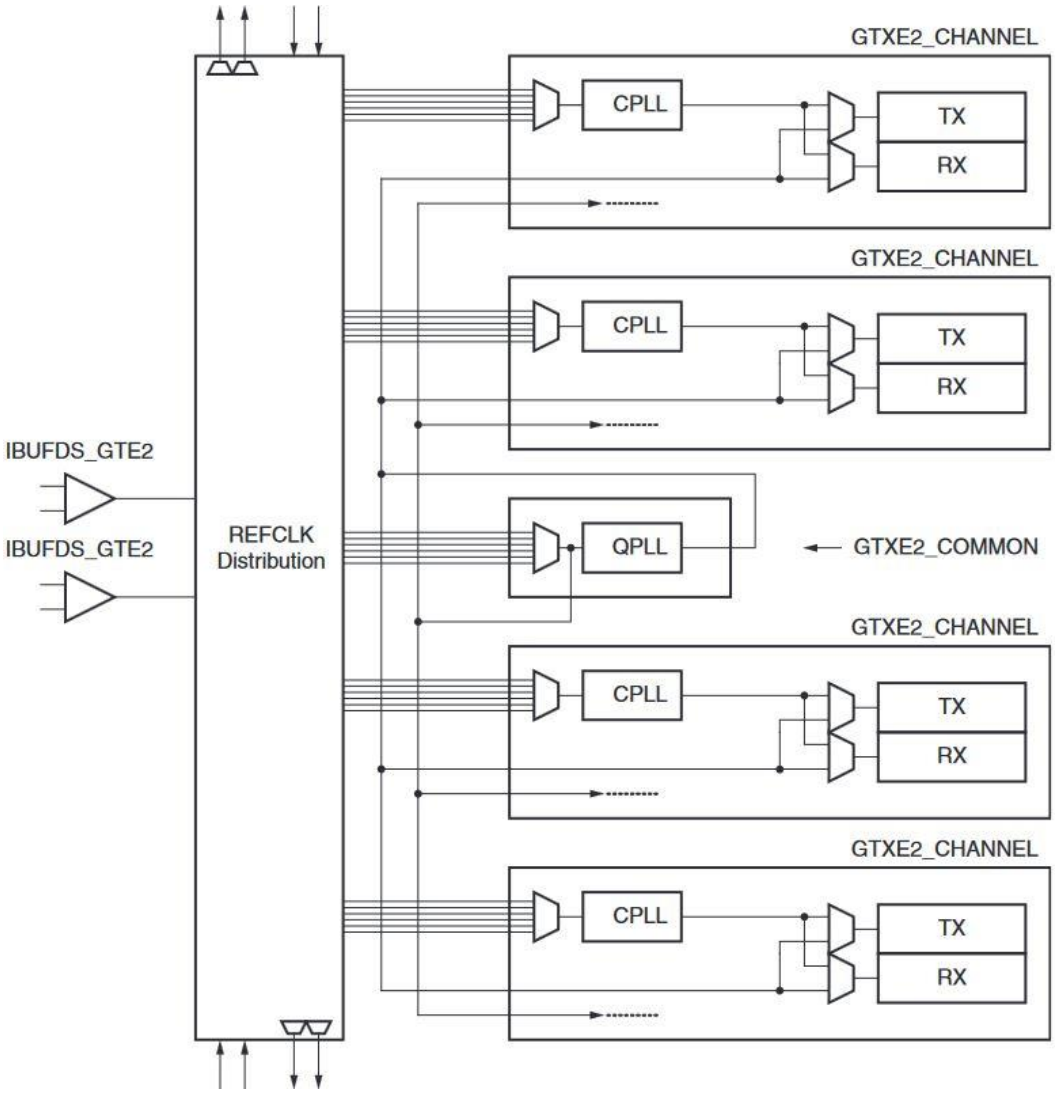


Figure 4.2: GTX Transceiver quad block diagram.

The selection of QPLL allows all four GTX channels to be synchronized, and reduces the total resources usage and power consumption of FPGA. The QPLL can be shared by four transceiver channels within the same quad, but cannot be shared by channels in other quads. The QPLL is an LC-tank PLL which has lower jitter characteristics and provides better performance in higher frequency band, the block diagram is shown in Figure 4.3. In the figure, the upper VCO band supports 9.8~12.5GHz and lower VCO band supports 5.93 to 8GHz. When the lower band VCO is selected, the upper band VCO is automatically powered down and vice versa. The input clock can be divided by a factor of M (reference divider) before it is fed into the phase frequency detector. The feedback divider N determines the VCO multiplication ratio. The QPLL output frequency (PLL CLKOUT) is half of the VCO frequency and applied to GTX transceivers. A lock indicator block compares the frequencies of the reference clock and the VCO feedback clock to determine if a frequency lock has been achieved. However, the QPLL is very limited to low-band VCO frequency and requires larger layout size and consumes more power per unit enabled. Therefore QPLL is more recommended to use as the line rate goes above 6.4Gbps (3.2GHz PLL output frequency) and multi-channel operation are enable.

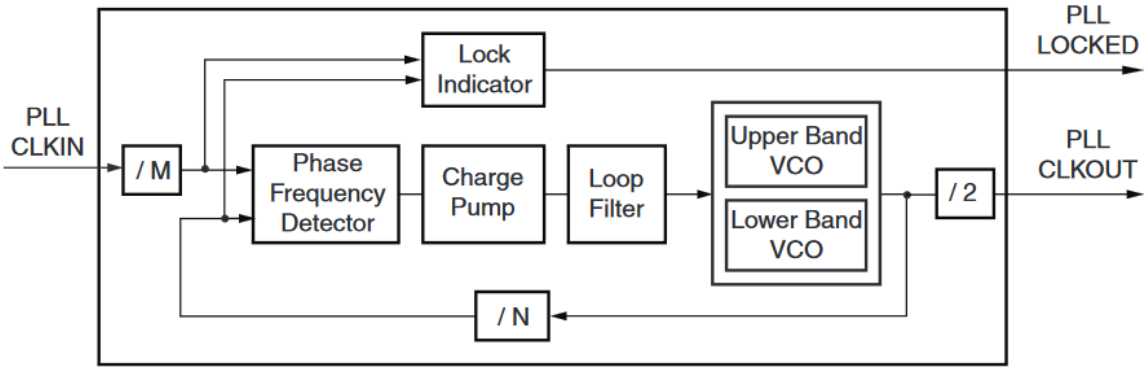


Figure 4.3: QPLL block diagram

On the other hand, if the applications require each GTX channel to work individually (not synchronized with other channels), then the CPLL is needed. Each GTX transceiver channel contains one ring-based CPLL and it cannot be shared by other channels even in the same quad. Figure 4.4 shows the block diagram of CPLL. Similar to

QPLL, the input clock can be divided by reference divider M before feeding into the phase frequency detector. The feedback dividers, N1 and N2, determine the VCO multiplication ratio and the CPLL output frequency. A lock indicator block compares the frequencies of the reference clock and the VCO feedback clock to determine if a frequency lock has been achieved. The CPLL in the GTX transceiver has a nominal operating range between 1.6 GHz to 3.3 GHz, and it has a better performance at lower line rate, wider band compatible and smaller unit size compares to QPLL. However, the total usage of four CPLLs with all four GTX channels enabled causes more power consumption then a QPLL with the same condition. In other words, if only a single GTX channel and a relative low line rate is needed, the CPLL is a more proper selection than QPLL.

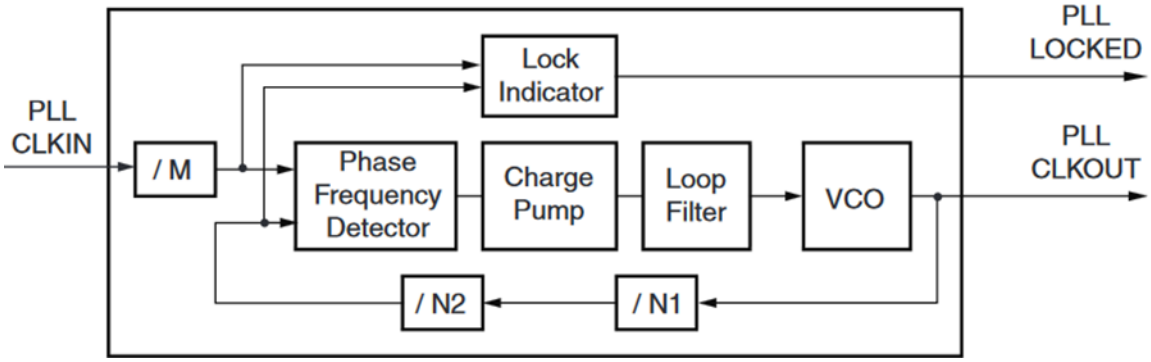


Figure 4.4: CPLL block diagram

Each GTX transceiver is composed of transmission block (TX) and receiving block (RX). Figure 4.5 shows the diagram of TX block of a single-channel GTX. As shown in the figure, the data flows from the right side to left side. The TX side includes two major blocks, which are TX physical coding sublayer (PCS) and TX physical medium attachment (PMA). The TX PCS processes the parallel TX data with low-speed parallel clock (TXUSRCLK), which is the fanout of reference clock distributed from global buffer. The very right-end of TX PCS is the FPGA TX interface, where the parallel-transmitted data and the configuration controls are supplied from. The TX data and control signals in this block is running under TXUSRCLK domain (typically below 500MHz). Users are allowed to define their own TX data pattern in this block, such as utilization of memory or coded-

pattern generators. The width of TX data (2-byte, 4-byte) is also defined in this block. The TX data is transmitted into GTX transceiver on the positive edge of TXUSRCLK. Also the configuration signals for GTX can be generated in FPGA TX interface. In other words, this block is the gateway between FPGA core logic and GTX transceivers.

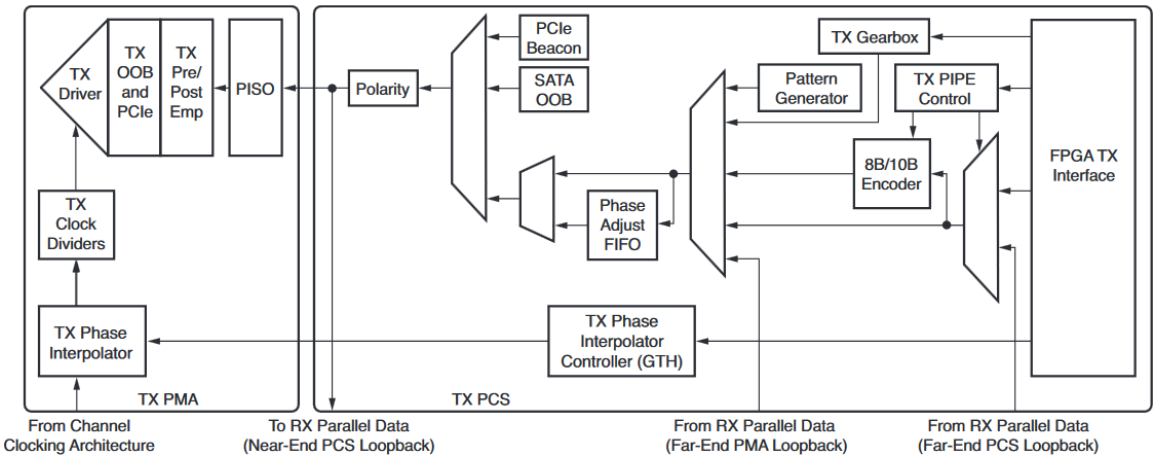


Figure 4.5: GTX TX block diagram

The TX data and configuration signals then are transmitted to user-selected data processing blocks. The TX data undergoes certain process while passing each block with the configuration signals and perform the function of these blocks. The TX PIPE Control provides the data path selection from either FPGA path or RX received parallel data path. RX received parallel data path is actually a loopback path which takes deserialized data from RX and queues it to TX side. Another loopback path in this block allows the parallel data signal to bypass TX PMA and transmit to RX side directly. 8B/10B is an industry standard encoding scheme that trades two bits overhead per byte for achieved DC-balance and bounded disparity to allow reasonable clock recovery. The GTX transceiver has a built-in 8B/10B TX path to encode TX data without consuming FPGA resources. Enabling the 8B/10B encoder increases latency, on the other hand it can also be bypassed to minimize latency. Some high-speed data rate protocols use 64B/66B encoding to reduce the overhead of 8B/10B encoding while retaining the benefits of an encoding scheme. Therefore TX gearbox provides 64B/66B and 64B/67B header and payload combining to extend the

supporting of those high-speed protocols. The TX pattern generator block generates several industry-standard pseudo random patterns including PRBS-7, PRBS-15, PRBS-23 and PRBS-31 which can be used to test the signal integrity of high-speed link. Also it includes the supporting of clocked pattern (square wave with 2 UI period), long-word pattern (square wave with 16 UI, 20 UI, 32UI, 40UI period) and PCI Express compliance pattern for other testing strategies. After passing all these blocks, the multiplexor in TX selects particular data path based on configuration setup generated from FPGA interface. Since there might exist skew within parallel TX data, the Phase Adjust FIFO block provides a fine phase adjustment for TX data if needed. In the end of TX PCS block, the data passes Polarity block which allow the user to sweep P-end or N-end of TX signals in case there is mistake of differential signal layout on PCB. Afterward the TX data is sent to TX PMA block for serialization and transmission.

The TX PMA block is basically composed of a high-performance serializer and a configurable driver. The PISO block (parallel in, serial out) uses the high-speed serial clock (typical several GHz) which is generated from either QPLL or CPLL to sample coming parallel TX data for serialization. TX driver is configurable, allowing differential voltage sweep, pre/post-emphasis control, PCIE features setup and out of band (OOB) control for serial at attachment (SATA) standard. These features help >3.2Gbps signal against resistance-capacitance (RC) effect and improve the high-speed signal quality. Also, the single-ended stream signal is transferring into differential signal then are supplied to TX pair pins on FPGA package in the analog end of TX driver.

The RX actually performs the opposite operation of TX side, which is shown in Figure 4.6. As the figure shows, the data flows from left to right side in RX block. Similar to TX block, each RX contains RX PMA and RX PCS block. High-speed serial data first flows from the pins on FPGA package into the RX PMA. The RX PMA is composed of a high-performance receiver, programmable equalizer, clock data recovery (CDR) and SIPO (serial in, parallel out) block. The configurable analog front end (AFE) is a current-mode

input differential buffer with the features of configurable RX termination voltage and calibrated termination resistors, which allows the receiver to receive signal of various protocols. The two programmable RX equalizers including decision feedback equalizer (DFE) and low-power mode (LPM) help CDR circuit to recover received high-speed data stream. These equalizers is selected depending on system level trade-offs between power and performance. For a lower-loss channel, LPM mode is chosen for power efficiency. On the other hand, for equalizing high-loss channels, the DFE mode is selected to offer better data-recovering quality. This recovered data stream from AFE then is sent to SIPO circuit and deserialized into parallel RX data.

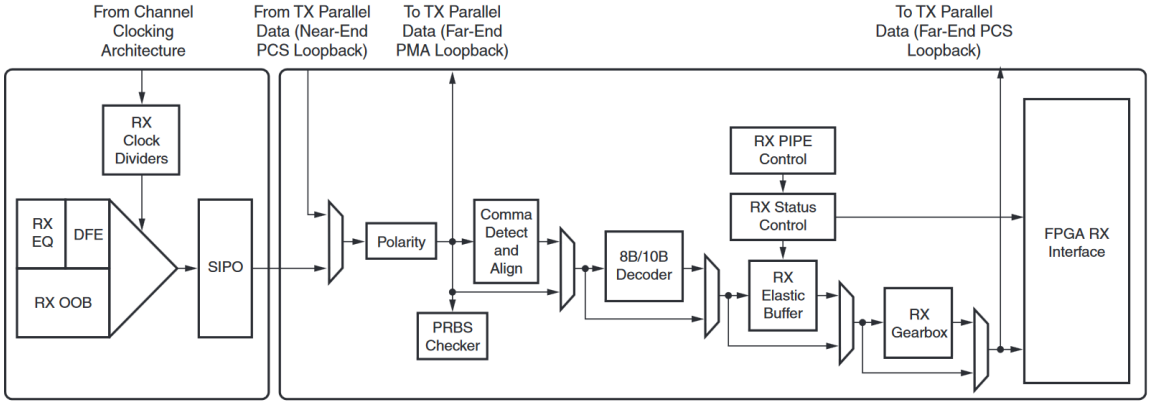


Figure 4.6: GTX RX block diagram

The parallel data then flows into RX PCS block. This RX data usually shares the same bit width with TX data which can be configured by FPGA TX/RX interface block. Comparing to TX PCS block, RX PCS also has the highly similar but opposite operation block such as polarity sweep control, 8B/10B decoder, RX gearbox for 64B/66B decoding and PRBS error checker to check bit error rate (BER). Also the comma detect is presented to align the input signal accordingly. These processes are under parallel clock RXUSRCLK domain which is the duplication of TXUSRCLK and also comes from the fanout of global reference clock. The RX PCS block also has built-in loopback paths to receive parallel TX data from TX PCS and the ability of transmitting received data to TX PCS. After passing this blocks, the processed parallel RX data is then provided to the FPGA RX interface.

4.2.2 Hardware system-level Design

The FPGA-based Testing Platform [49] was mainly built by using Xilinx Kingtex-7 FPGA as the main controller, combing with auxiliary chips including oscillator, flash memory and USB UART, USB communication port, JTAG programming port, general purpose I/O (GPIO) SMA ports, and high-bandwidth connectors. This board is designed to either plug-in on ATE as extension card to be controlled by ATE host, or operate individually by using communication interfaces to program and control. The power can also be supplied by either ATE or external power supplies.

Figure 4.7 shows the block diagram of FPGA-based Testing Platform. This board contains 4 I/O extension slots, each of the 4 slots is soldered on with a 10GHz high-bandwidth connector. These connectors are used to transmit wide-band (DC to 10Gbps) and multi-channel (up to 100 channels) signals between the FPGA and custom-designed extension board. Each extension slot supports 4 complete bi-directional GTX channels (differential, total 16 pins are occupied), so that total 16 high-speed channels can be implemented on this testing platform. There are 64 channels on each slot can be applied to moderate-speed (below 1Gbps) testing or implemented as control ports or communication I/Os such as serial peripheral interface (SPI) which is widely used to program analog/digital devices. Eight independent power pins on the connector are separated from those signal pins to distribute the regulated voltage supplies form the main board to power on extension boards. Besides the high-speed connectors, we also connect four moderate-speed I/Os to SMA connectors for system debug. With these general purpose I/O (GPIO) ports, the signals inside FPGA can be pulled out directly and measured by oscilloscopes or spectrum analyzers, which is convenient for users to fix and improve this system.

Two reference clock sources are provided on this board. The first one is a 200MHz oscillator with LVDS outputs. Since most FPGA logics are able to operate at this speed and the frequency of GTX reference clock also falls at this range (<500MHz), this oscillator is an appropriate clock source for FPGA. However, the drawback of using soldered

oscillator is it only outputs fixed or limited rate of clock and lacks flexibility. Therefore an external reference clock is provided by differential SMA connectors on this board. By the help of these SMA connectors, the FPGA can use more flexible clock frequencies from signal generators for different testing strategies.

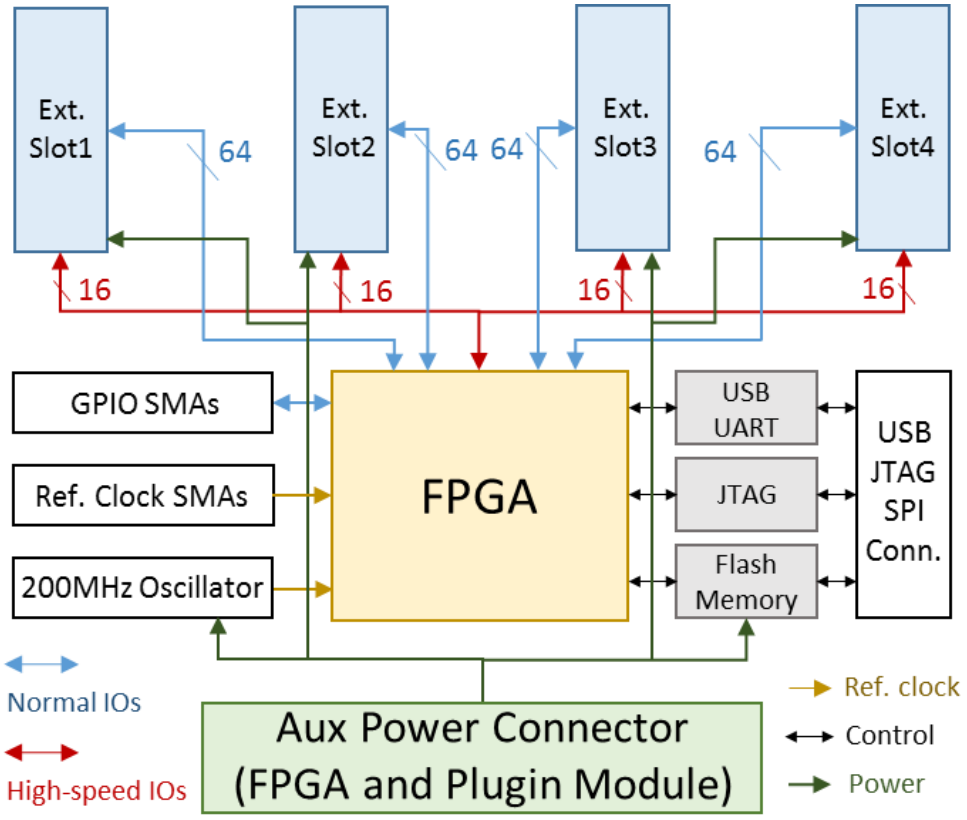


Figure 4.7: FPGA-based Testing Platform block diagram.

Several I/O interfaces are built for the FPGA to communicate with user interface, auxiliary chips and future extension board. JTAG is the main interface to program or communicate FPGA, by linking this port with user's computer, the FPGA firmware can be reprogrammed for different purposes. In this research we also use JTAG to configure the FPGA to simplify the complexity of the firmware and reduce I/O consumption. The USB interface is a widely-used standard in the industry and many communication protocols are implemented base on it, therefore we also reserve the support of USB UART IC and connector on this board. Since FPGA is a volatile component and it loses the content of bit-stream firmware once the power is off, a flash memory is built to store firmware file.

When we power this platform on, the FPGA will be automatically reprogrammed with the stored bit-stream file without manually programing procedure by the computer. Therefore the real “independent” operation for this main board is thus realized.

The photo of this platform is shown in Figure 4.8. On this prototype board, only one Kintex-7 FPGA and 4 extension connectors have been implemented. However, this board has reserved enough space for the other three FPGA chips and 12 connectors for future expansion. With a re-arrangement of layout, potentially there can be up to four FPGA chips and offer 16 extension slots on this design. According to what we have addressed before, each extension slot is connected to four high-speed GTX channels so the entire system can potentially offer up to 64 10Gbps channels for testing.

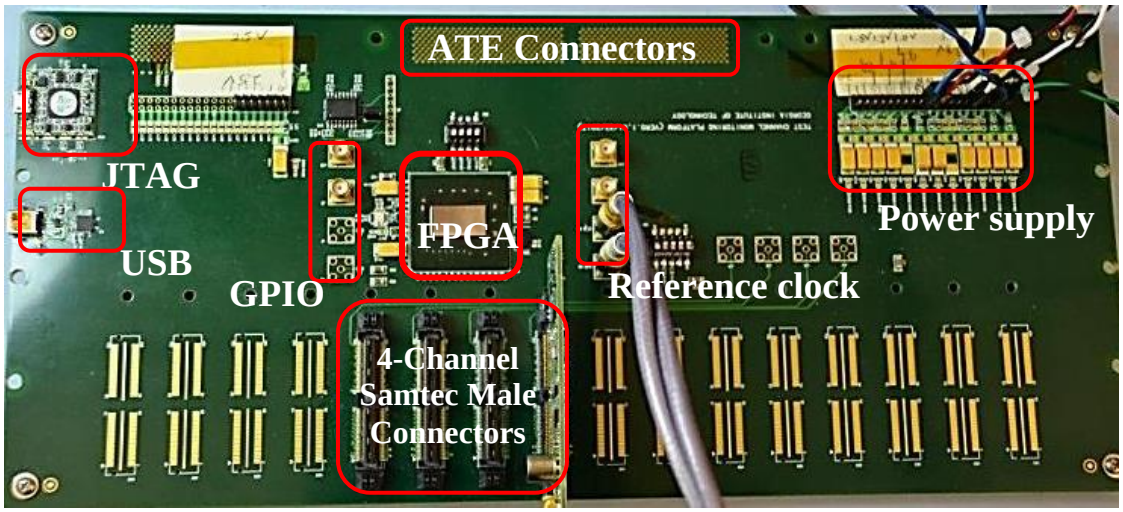


Figure 4.8: The actual photograph of FPGA Main board.

On the top of the Figure 4.8, a special connecter is reserved to be host by ATE systems. The board is also designed to be the interface between ATE and DUT board, which is shown in Figure 4.9. The host ATE provides power supply, transmit data to the FPGA board and receive data from FPGA. The ATE actually utilizes this FPGA board to update new testing features for DUT (similar to the ATG in last chapter). The main power supply is located at top right corner in Figure 4.8. Since the FPGA supports various I/O standards, the DC offset and biasing voltages from these standards are also provided on the board. Despite of 1.0V core voltage, the power plans include most of industry used I/O

voltage such as 1.2V, 1.8V, 2.5V and 3.3V. Furthermore, the power plan on the main board also reserves potential power supplies requirement for the extension board. In order to provide the power to the extension board, the power plans on this platform are connected with the separated power pins on the high-bandwidth connectors.

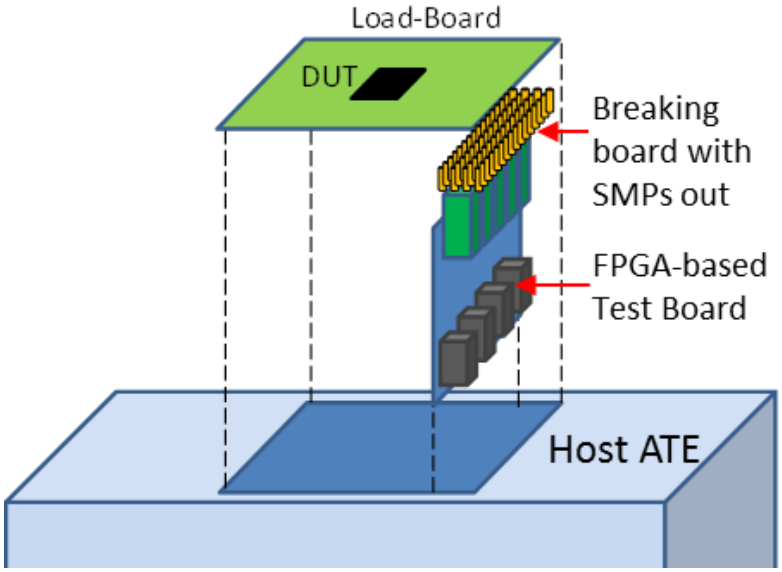


Figure 4.9: FPGA-based Testing Platform on host ATE.

4.2.3 FPGA Block Design

In order to configure this testing platform to wide-band testing applications, a separated low/high-speed architecture is selected to be implemented inside the FPGA. Figure 4.10 illustrates the top level of design block we have developed for the general-purpose testing. The logic design of this architecture is implemented by Verilog hardware description language and compiled by Xilinx ISE 14.0 software. The bit-stream file (firmware of FPGA) generated from HDL code is upgradable and reconfigurable inside FPGA, hence the new features can be added easily by reprogramming the updated firmware. Furthermore, since this FPGA-based platform might be applied to host different plug-in applications, the configuration logics and I/Os required for controlling these extension modules can also be implemented by updating the existing HDL code.

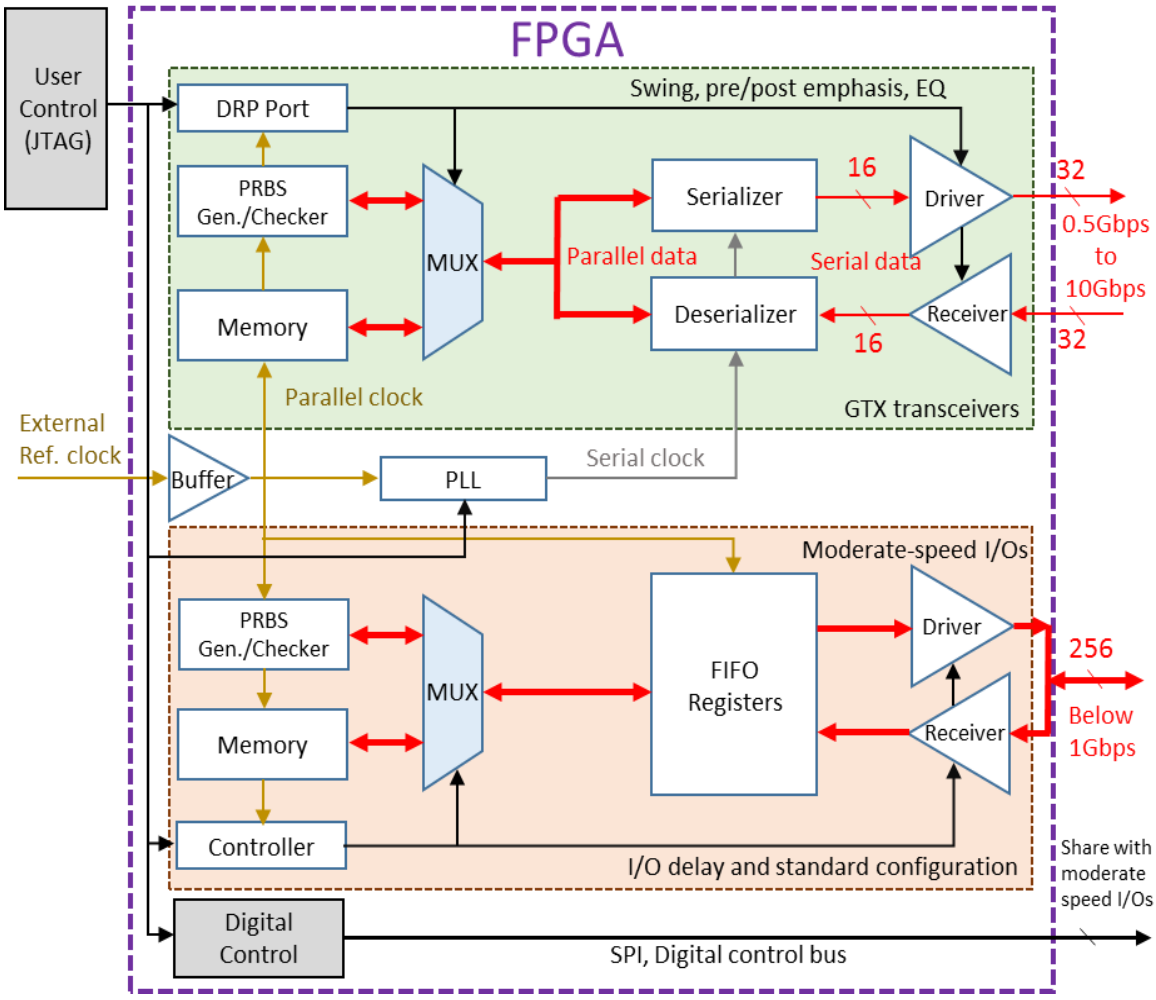


Figure 4.10: Testing architecture within the FPGA

For low-speed I/O testing (typically <1Gbps) such as USB2.0 (480Mbps) or PCI (533Mbps) standard, the FPGA moderate-speed I/Os (bottom block of Figure 4.10) are applied to transmit and receive the signals. The reference clock samples the data in first-in first-out (FIFO) registers and output the signal on both positive and negative edge (DDR technology). Therefore the output data rate is twice of the reference clock frequency. The sampling FIFO access the data from a data path composed of various data pattern generators and a memory system. PRBS pattern generator is the most widely used pattern for testing I/O performance, therefore it is code-realized within the FPGA firmware design. In addition to the random data generator, refreshable memories are available and allow users to define and store the specific patterns they need for the DUT. The low-speed FPGA

I/O drivers can be programmed to either single-ended or differential I/O standards [47] with appropriate voltage swing and DC offset. The delay and slew rate [47] of each I/O is also programmable. A digital controller block which can be accessed by user through JTAG port outside of FPGA is built to configure these ports.

If the several Gbps signal is needed for the testing, such as USB3.0 (5.0Gbps) and PCIE (8Gbps) standard, then the high-speed GTX transceivers are selected (top block of Figure 4.10). Since GTX itself is a complicated hardware IP, it is designed to be accessed by an individual dynamic reconfiguration port (DRP). The DRP control block is built inside FPGA TX/RX interface and allows user to communicate with GTX transceivers. This DRP port can also be configured by JTAG port outside of FPGA. Several critical configurations for GTX transceiver are to define parallel data bit-width, parameters of generating high-speed serializing clock and TX driver settings. The parallel data bit-width can be configured to 16, 32 and 64bits or 20, 40 and 80bits with 8B/10B encoding/decoding is enable, a wider bit-width is more recommended at higher data rate (>6.4Gbps) since the parallel clock need be limited under 670MHz (the maximum reference clock of this Kintex-7 FPGA) [79]. The high-speed serializing clock used to determine the output line rate is generated by appropriate PLL settings. QPLL is known to have better performance at high band and therefore is considered to be a better solution over CPLL. The formula to determine the QPLL output clock frequency is shown in Equation 4.1:

$$f_{PLL\_clk} = f_{ref} \times \frac{N}{M \times 2} \qquad \text{Equation 4.1}$$

M is the reference divider and N is feedback divider, the available values for M and N is shown in Table 4.2. These two parameters control voltage control oscillator (VCO) to generate high-frequency serial clock. With a fixed reference clock, the VCO frequency of QPLL clock can be defined by different combinations of the values of M and N. Furthermore, there is a hardware built-in divided-by-2 circuit after VCO in QPLL, so the frequency of QPLL output clock is half of the VCO frequency.

Table 4.2: PLL Divider Settings

QPLL		CPLL	
Factor	**Value**	**Factor**	**Value**
M	1, 2, 3, 4	M	1, 2
N	16, 20, 32, 40, 64, 66, 80, 100	N1	4, 5
		N2	1, 2, 3, 4, 5
D	1, 2, 4, 8, 16	D	1, 2, 4, 8

The final data rate of output is defined by the Equation 4.2. The parameter D is post divider which can be manually programmed to output low frequencies if needed. The VCO is usually operating at relative high frequency (several GHz) to achieve the best jitter performance. If we want to generate low data rate signal and change to low-speed I/O is not applicable, the ideal solution is to operate the VCO at typical operation frequency and use post divider to divide the PLL output frequency down. The available values of post divider "D" are also shown in Table 4.2. Typically it is set to "1" to achieve full speed of GTX. A simple example is given to describe this process: if there is a 200MHz reference clock with the settings of M =1, N = 32 and D = 8, we are able to generate an 800Mbps output signal. The combinational setups of these dividers (M, N, and D) allow a wide range of data rates with a fix reference clock.

$$f_{DataRate} = \frac{f_{PLL\_clk} \times 2}{D} \qquad \text{Equation 4.2}$$

The CPLL is an alternative option to generate serializing clock in single-channel application with lower line rate. It follows the similar process of generating serializing clock, the formula of determining the frequency of CPLL output clock is shown in Equation 4.3. Note here the N1 and N2 are feedback divider sets and there is no "divided-by-two" circuit in CPLL, the values of CPLL parameters can also be referenced in Table 4.2. The

post divider and other setup is very similar to QPLL, therefore the final line rate of using CPLL can also be referenced in Equation 4.2.

$$f_{PLL\_clk} = f_{ref} \times \frac{N1 \times N2}{M} \qquad \text{Equation 4.3}$$

Not only for those critical bit-width and PLL configurations in GTX transceivers, the DRP is also used to configure TX driver which provide a 4-bit (16-step) differential voltage swing (peak to peak) from 270mV to 1100mV, and a 32-step pre/post-emphasis configuration is also provided to compensate the signal distortion of giga-bit signal transmission. On the RX side, the DRP is able to communicate with equalizer options including DFE and LPM block to optimize the recovered bit stream. Furthermore, the DRP is able to configure the TX/RX PCS blocks such as 8B/10B encoder/decoder, both PRBS generator/checker, TX/RX Gearbox, loopback path control, data alignment, the moderate-speed I/O configuration and even the controls for other customized logic block can be accessed by DRP port. Since the DRP is able to access these parameters dynamically, it is very convenient for signal quality adjustment without stopping the whole test to reconfigure these settings.

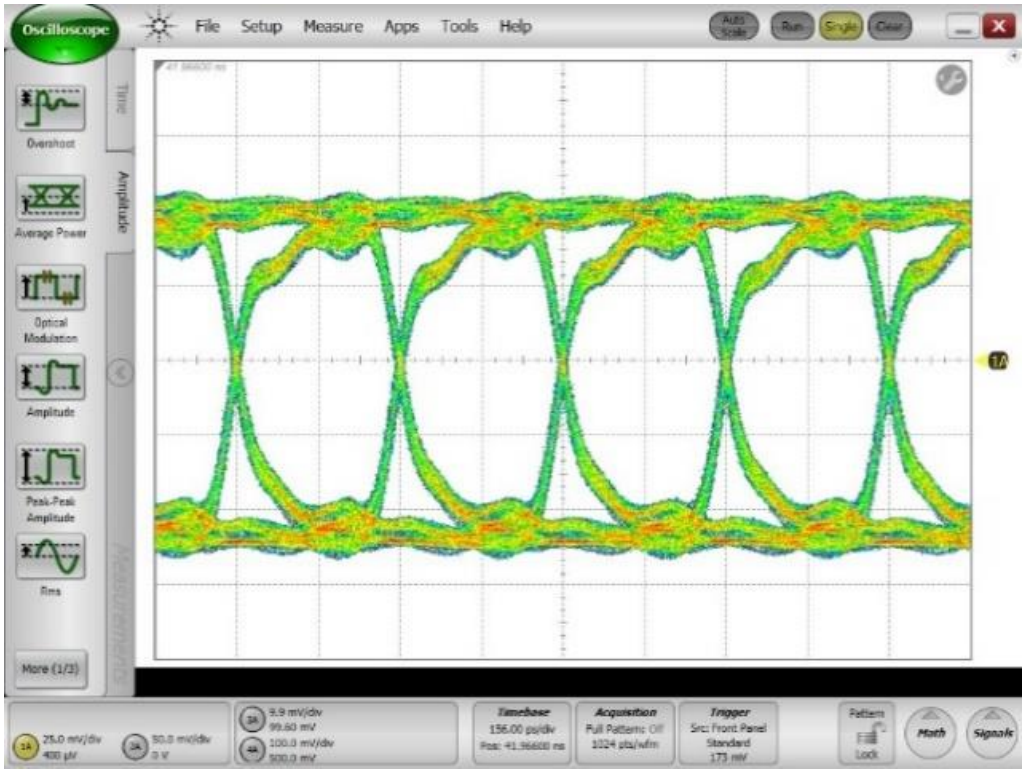
Despite the DRP and control unit for the configuration, another controller (bottom-left of Figure 4.10) is implemented to generate control signals for the devices on the main board or future plug-in board. As the core of the testing platform, the FPGA is able to host the DC control pins (only low current required) and be used to implement the communication interface (i.e. SPI) between the FPGA and other devices. Note that the control bus and I/O interfaces share the same resource of moderate-speed I/O channels. Therefore if these I/Os are used for control I/Os, the testing platform loses the same amount of moderate-speed testing channels simultaneously. In this chapter, we had only implemented DRP and JTAG for FPGA-based Test Platform, the reserved controller (i.e. SPI) for plug-in boards will be discussed individually in Chapter 5.

4.3 Experiments and Testing Results

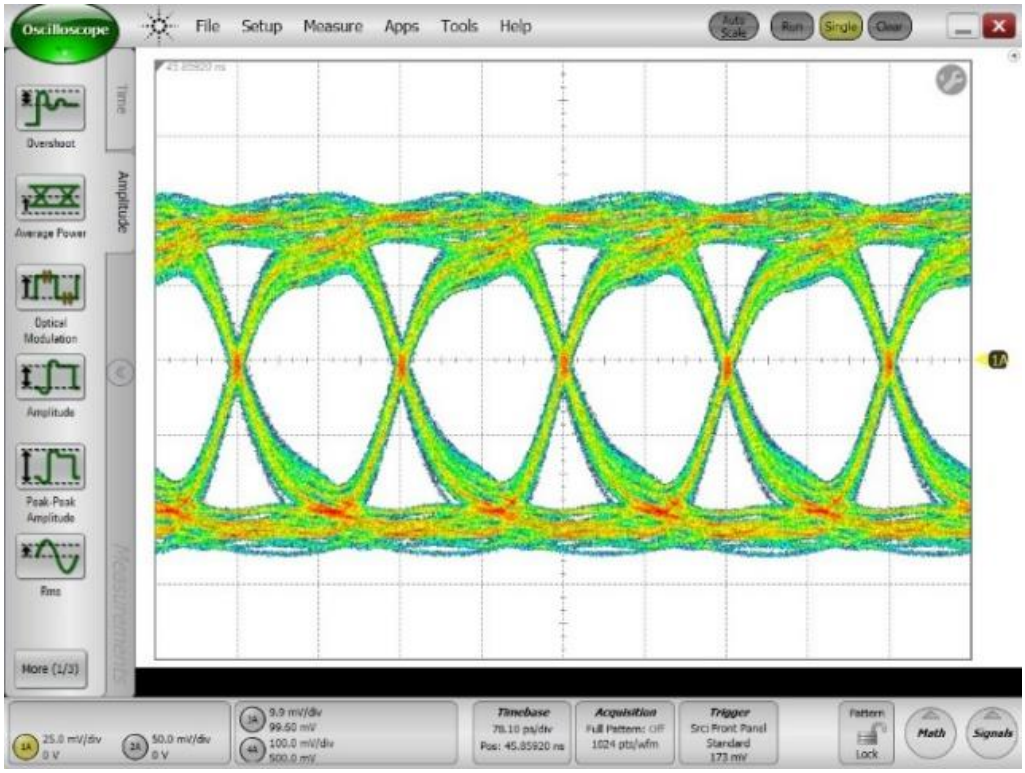
In this section we will demonstrate the performance of FPGA-based Testing Platform. Since the main application of this platform is high-speed transmission, the experiments will focus more on the performance of GTX transceivers. To obtain the best jitter performance, the FPGA main board uses the 200MHz reference clock form Keysight 81133A pattern generator. This generator is able to produce clocked, PRBS and limited user-defined data with only ~1.5ps RMS jitter. The SERDES bit-width are configured and fix to 32-bit wide since the data rate might go up to 10Gbps. Based on the knowledge we have so far, we are able the obtain 6.4Gbps signal form those settings. The QPLL is selected due to the better performance at high band and the multi-channel ability. All the measurements collected in this section is by a Keysight 86100D scope with 54752A 50GHz testing module. The triggering signal of the scope is also provided by the 81133A pattern generator to achieve the best results.

4.3.1 Performance

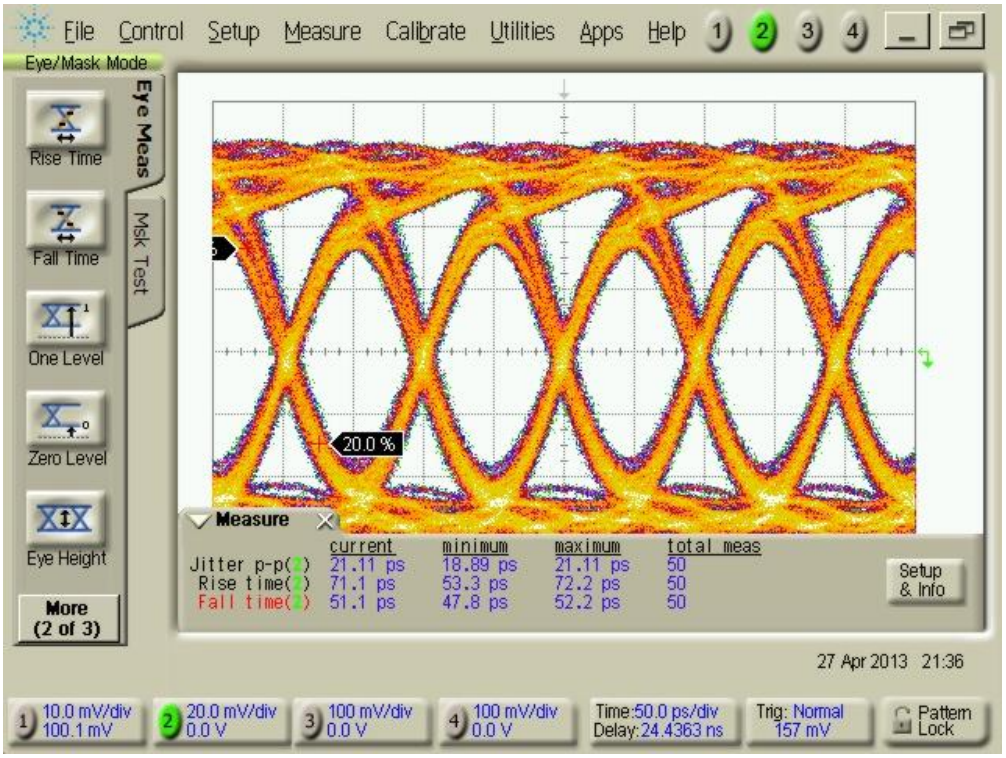
First of all we will demonstrate the performance of this FPGA testing platform. The data rate change is based on the various reference or VCO clock frequencies (in this experiment we fixed reference divider and feedback divider) and the value of post divider. We programmed GTX transmitters to produce high-quality signals at 3.2Gbps (the typical speed for most ATE systems) in Figure 4.11(a), then try higher performance at 6.4Gbps in Figure 4.11(b), 10.0Gbps in Figure 4.11(b) to show that this platform can be applied to some fastest standards in the industry. Also we had pushed the performance of the system to even as high as 12.0Gbps and is demonstrated in Figure 4.11(d). These figures show the maximum speed that this platform is able to achieve and the nice quality of the signal (~20ps peak to peak jitter, ~40ps rising/falling time) with clear and continuous opening eyes. These measurements also ensure testability when we apply this platform to test higher performance applications.



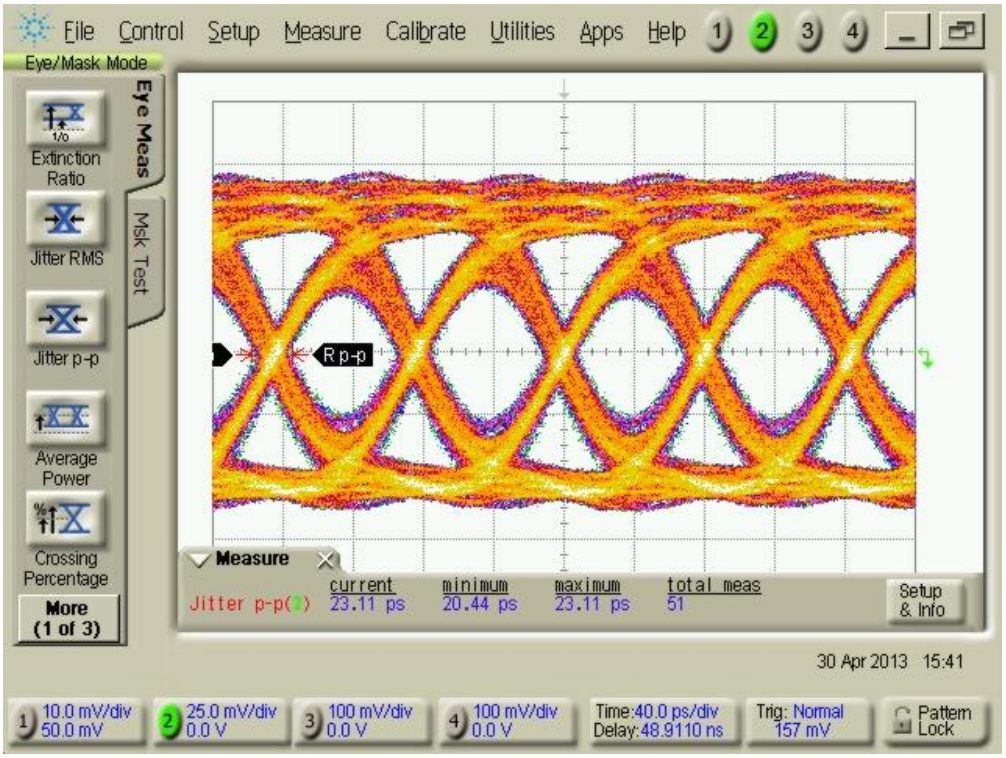
(a) 3.2Gbps single-ended PRBS eye diagram, 156ps/div and 500mV p-p swing.



(b) 6.4Gbps single-ended PRBS eye diagram, 78ps/div and 500mV p-p swing.



(c) 10Gbps single-ended PRBS eye diagram, 50ps/div and 500mV p-p swing.



(d) 12Gbps single-ended PRBS eye diagram, 40ps/div and 500mV p-p swing.

Figure 4.11: FPGA-based Test Board performance at different data rate.

4.3.2 Voltage Swing Control

The TX amplitude swing is programmable for GTX transceivers. Four-bit binary code controls peak to peak voltage swing from 140mV to 600mV single-ended, or 270mV to 1100mV differential. The amplitude control is about 25mV per step, and the comparison of actual measurement with the specification is show in Figure 4.12. In the figure, we can see the trend of the actual result is following the prediction. The data rate is fixed at 3.2Gbps with 32-bit SERDES mode and PRBS-31pattern in the measurement. All the signal is measured at single-ended and the result is shown in Figure 4.13. We first programed minimum voltage swing in Figure 4.13(a), followed with a half swing and three-quarter swing in Figure 4.13(b) and Figure 4.13(c) respectively, and in the final the drivers was programmed to maximum swing in Figure 4.13(d). The figures show excellent signal quality over different amplitude settings.

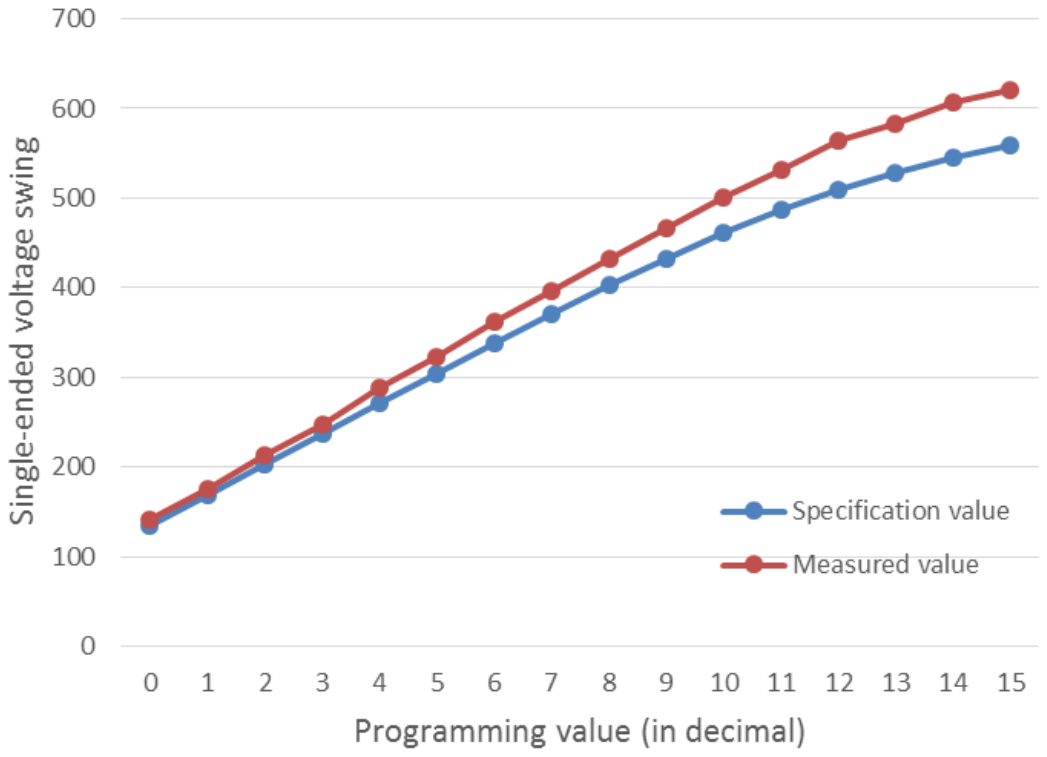
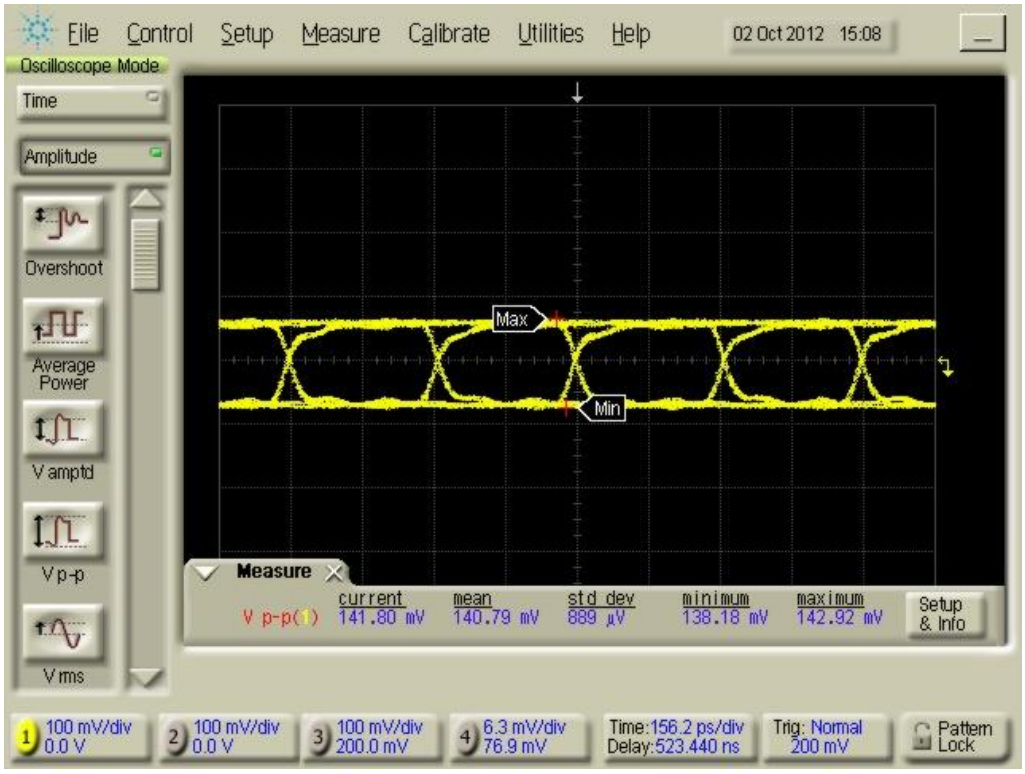
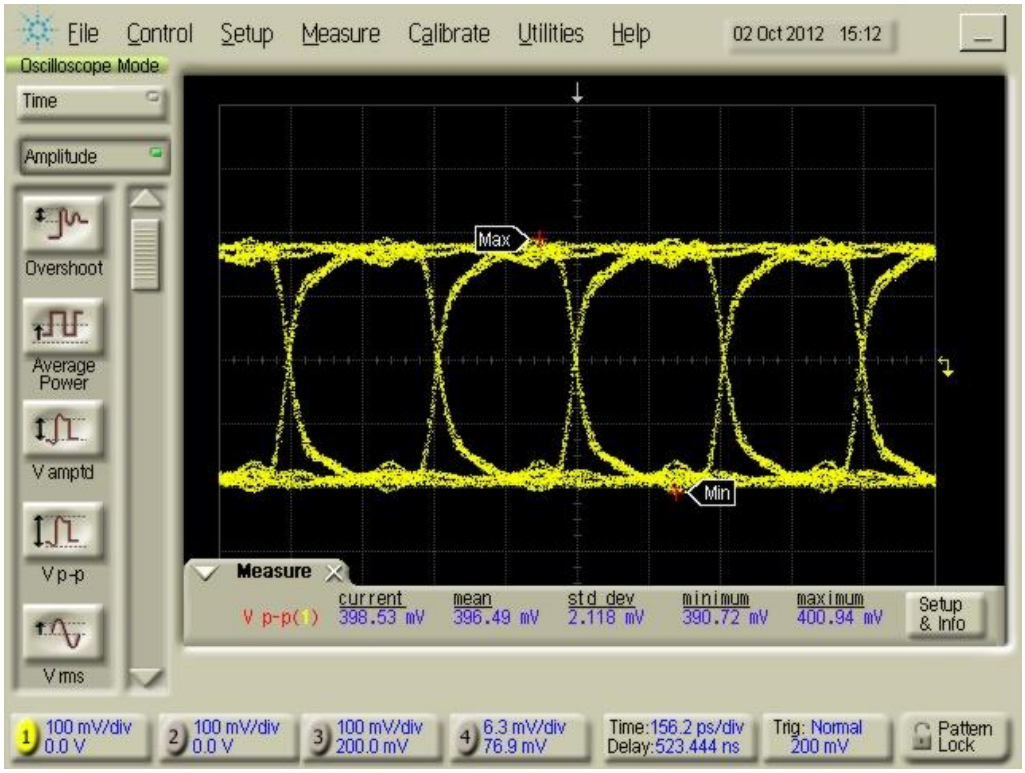


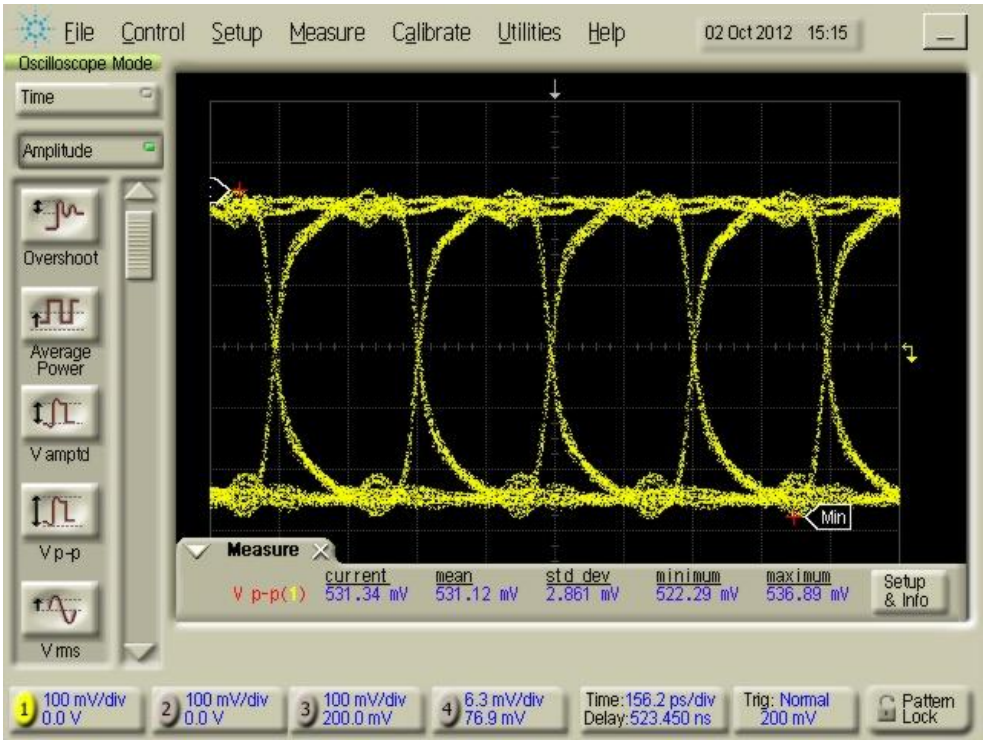
Figure 4.12: The comparison of measured voltage swing and specification.



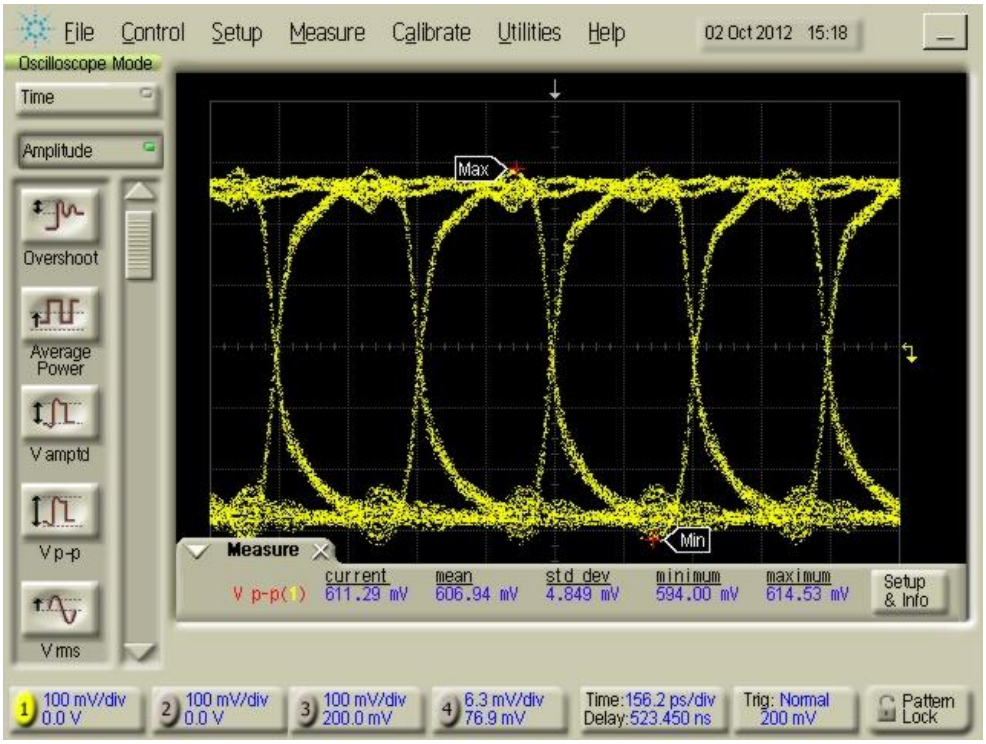
(a) Amplitude control code ”0000” with 140mV single-ended p-p swing.



(b) Amplitude control code ”0111” with 400mV single-ended p-p swing.



(c) Amplitude control code ”1011” with 530mV single-ended p-p swing.

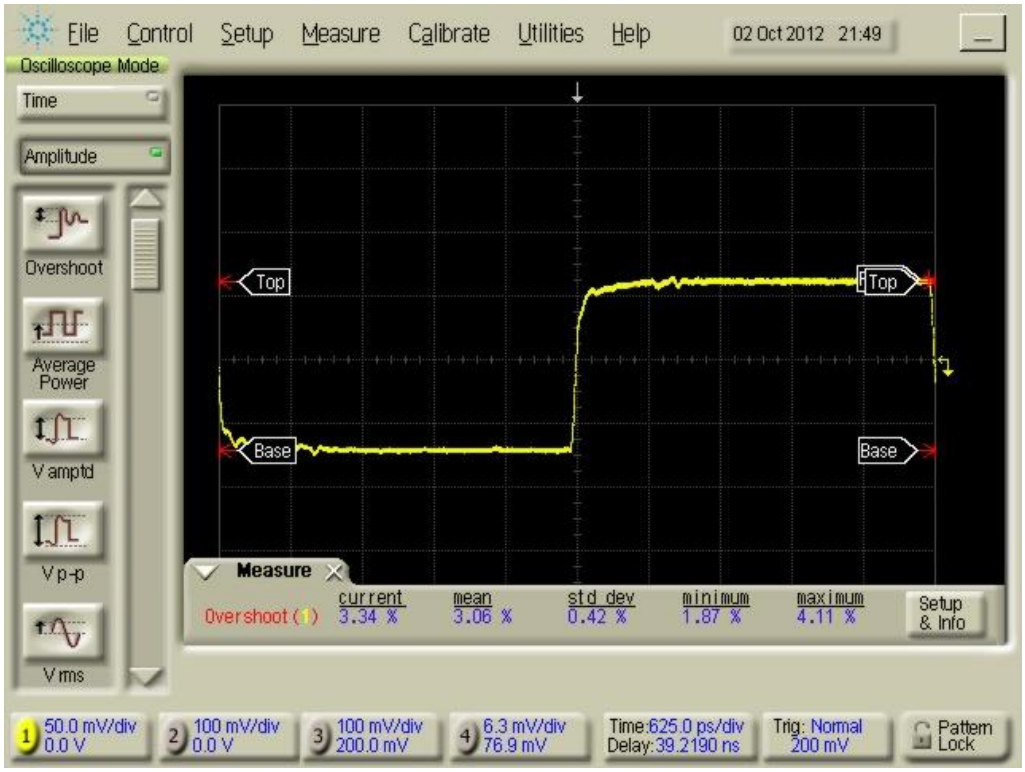


(d) Amplitude control code ”1111” with 600mV single-ended p-p swing.

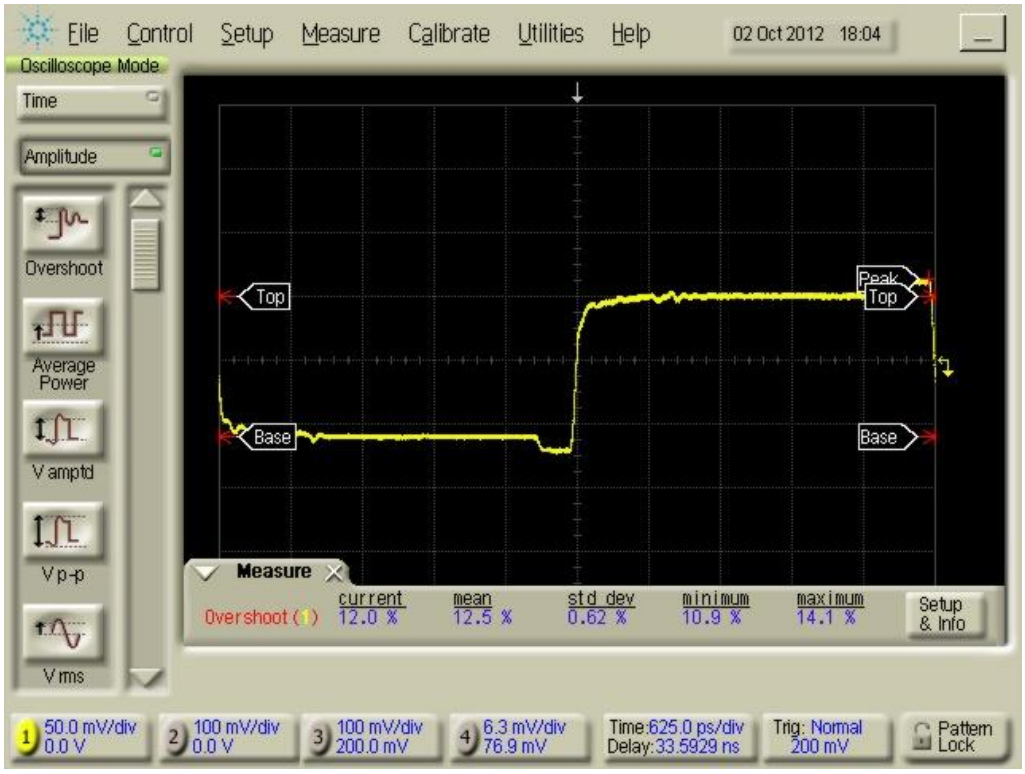
Figure 4.13: Amplitude control measurement at 3.2Gbps, PRBS-31 single-ended signal. Time based is 156ps/div.

4.3.3 Pre/Post-emphasis demonstration

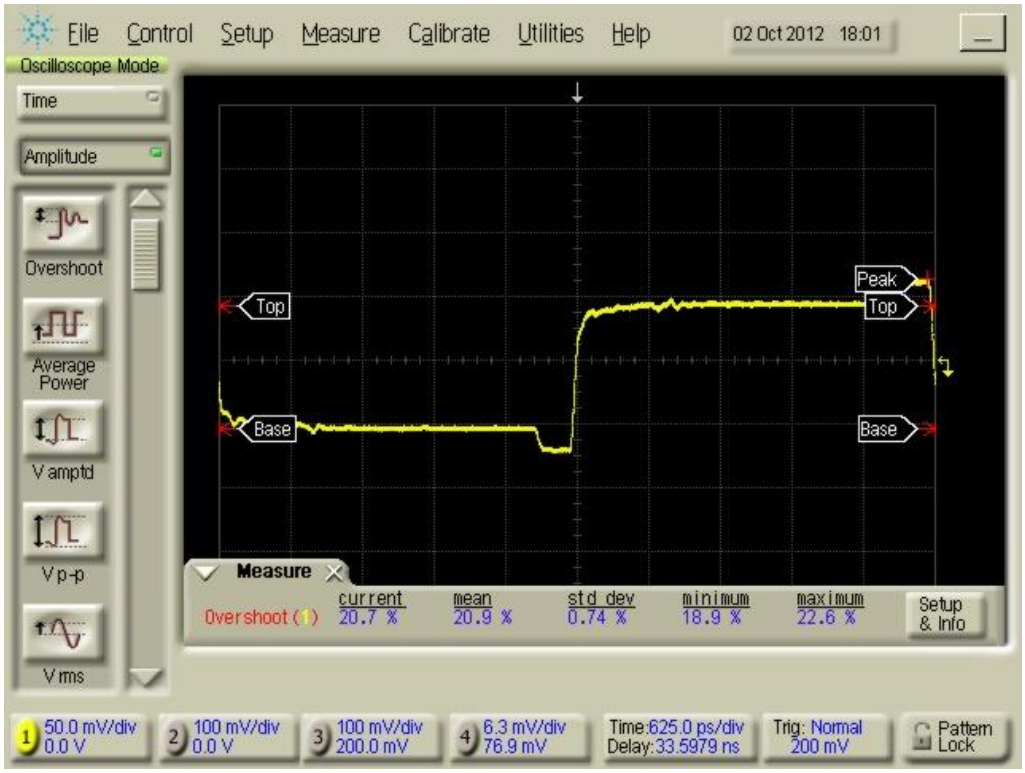
The pre/post-emphasis helps the signal to against the signal distortion from the physical world. In this section, we programed the FPGA to fix at 3.2Gbps. 32-bit SERDES mode and long pattern (16 “0”s 16 “1”s) was selected since practically longer pattern is easier to observe the emphasis effect on the scope. The overshoot ratio of the pre/post-emphasis was calculated by $(V_{peak}-V_{top})/V_{top}$. Figure 4.14 shows the measurement of no emphasis (a), half emphasis (b), three-quarter emphasis (c) and maximum emphasis (c) in 16-step pre-emphasis control. The overshoot can be observed at the beginning of rising/falling edge (where marks “Peak”), and the maximum overshoot of pre-emphasis is ~30%. The measurement of 32-step post-emphasis control is shown in Figure 4.15, which also follows the sequence of no emphasis (a), half emphasis (b), three-quarter emphasis (c) and maximum emphasis (c). The overshoot can be observed at the end rising/falling edge, and the maximum overshoot of post-emphasis is about 170%.



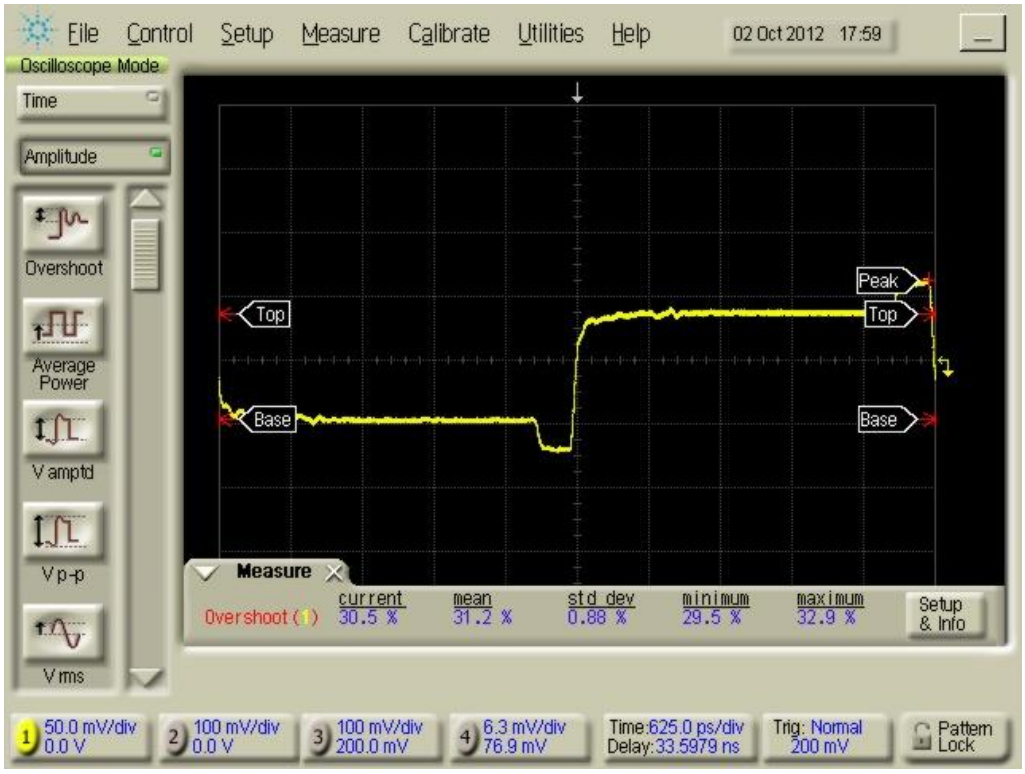
(a) Pre-emphasis control code ”0000”, with 0% overshoot.



(b) Pre-emphasis control code ”0111”, with 12% overshoot.

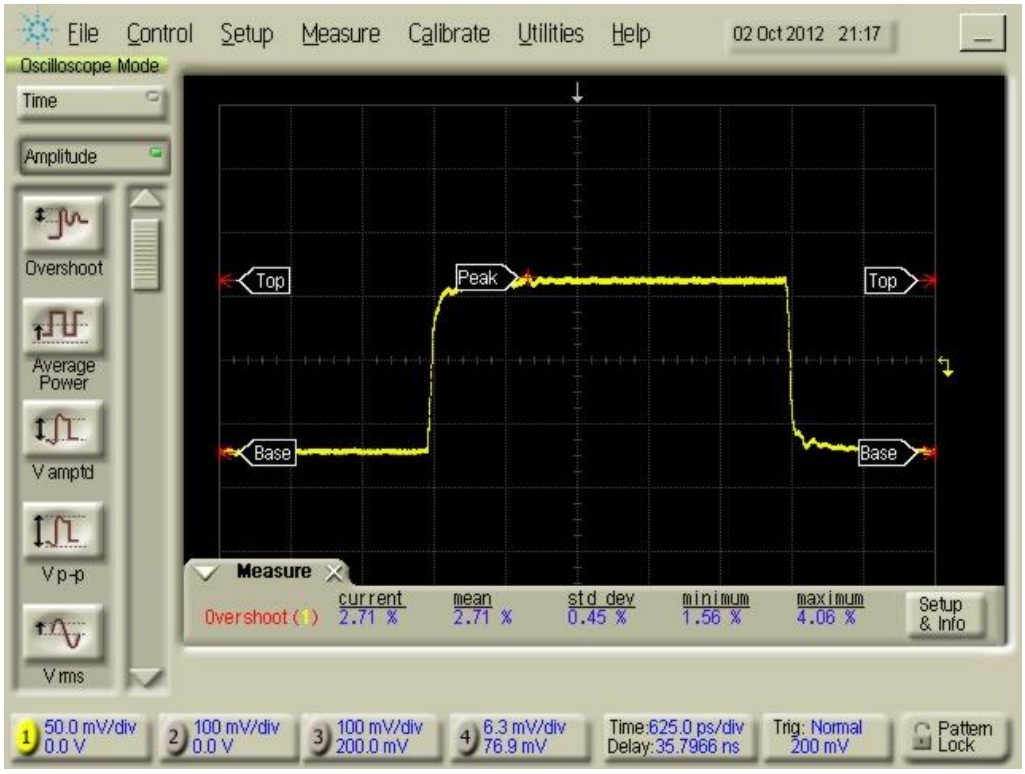


(c) Pre-emphasis control code ”1011”, with 21% overshoot.

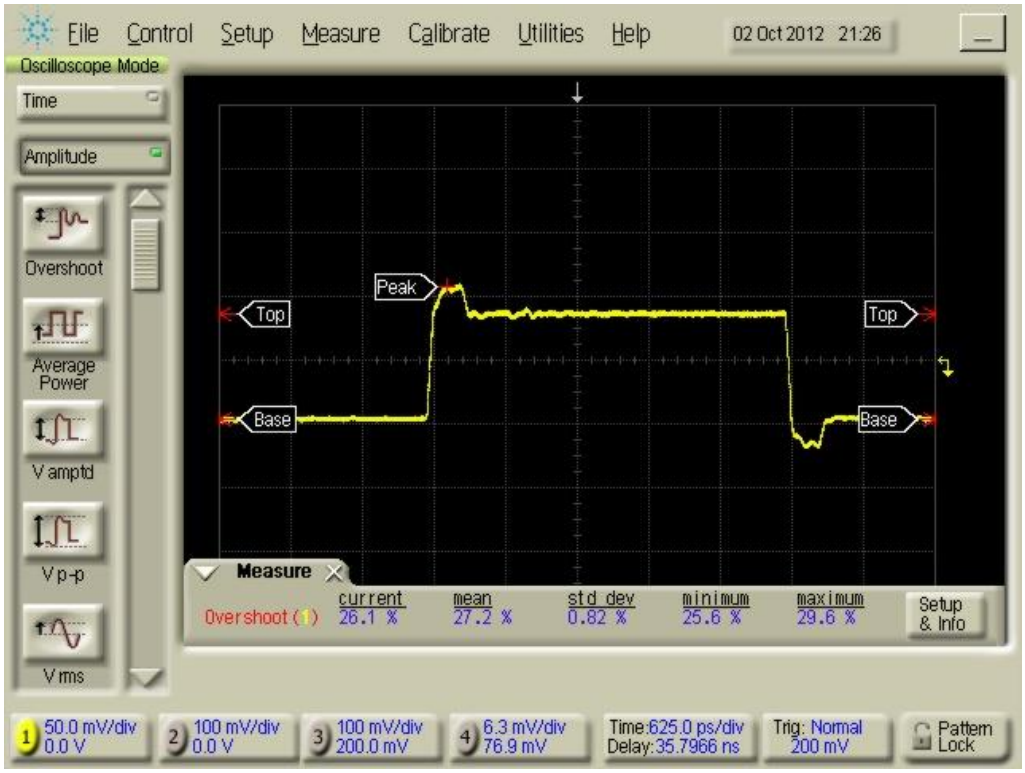


(d) Pre-emphasis control code ”1111”, with 31% overshoot.

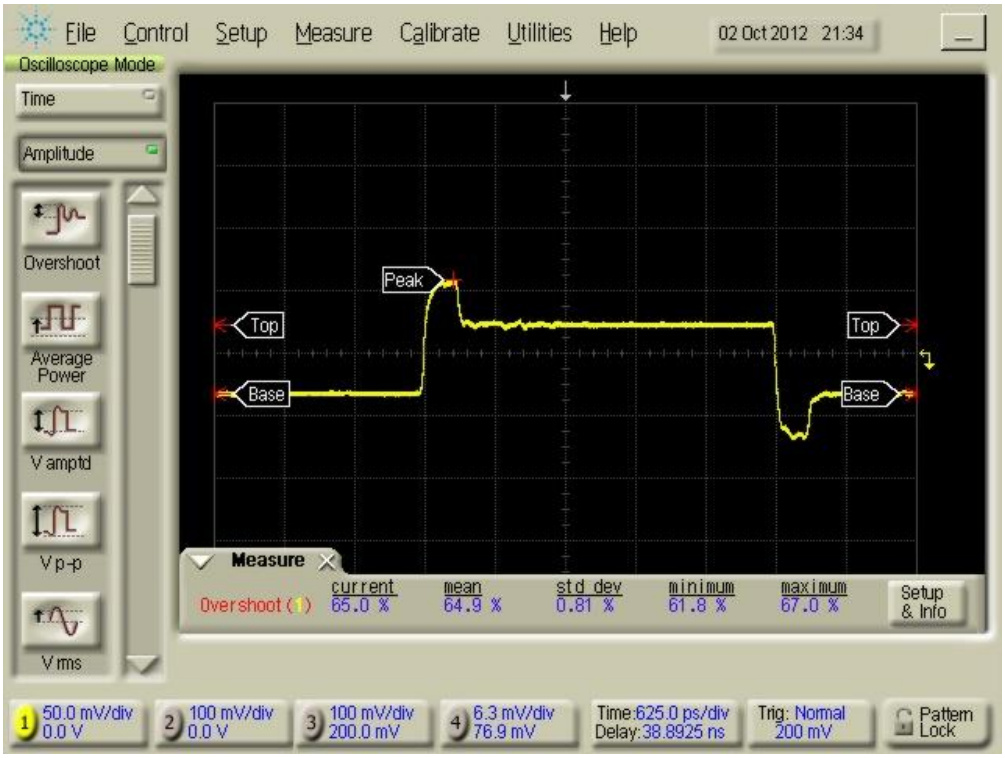
Figure 4.14: Pre-emphasis overshoot ratio measurement of Kintex-7 FPGA



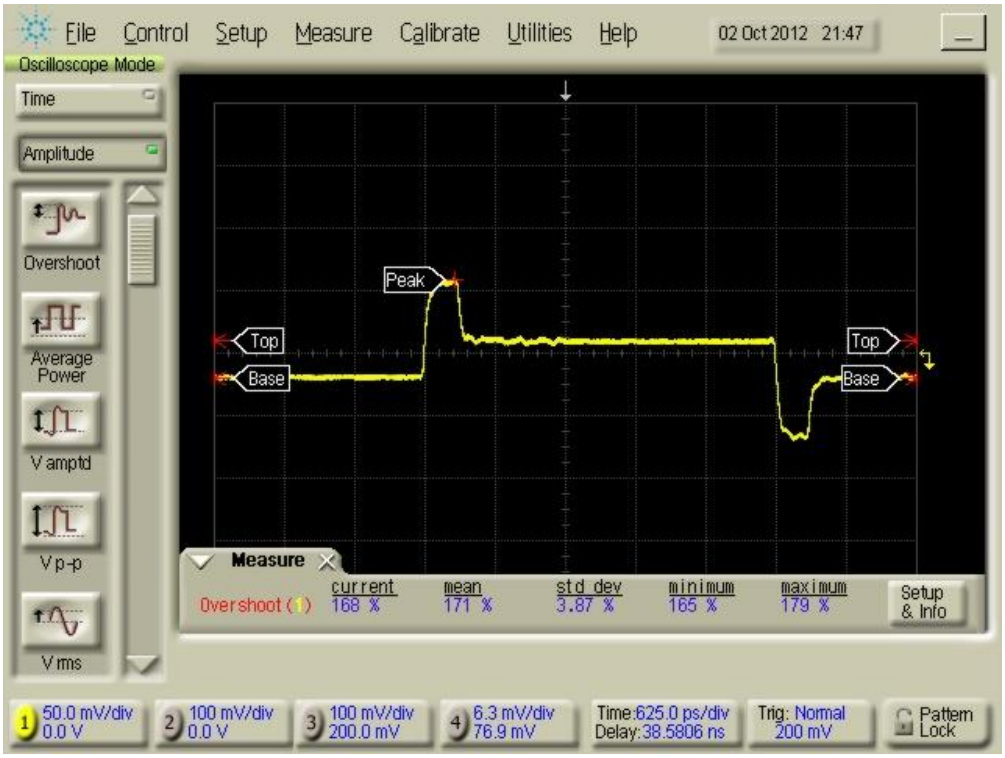
(a) Post-emphasis control code ”00000”, with 0% overshoot.



(b) Post-emphasis control code ”01111”, with 27% overshoot.



(c) Post-emphasis control code ”10111”, with 65% overshoot.



(d) Post-emphasis control code ”11111”, with 171% overshoot.

Figure 4.15: Post-emphasis overshoot ratio measurement of Kintex-7 FPGA

4.3.4 16Gbps Single-channel demonstration

The GTX of Kintex-7 is expected to run at 10Gbps according to the specification [79]. However, we still want to know the limit of the current FPGA and try to push the speed to a higher level. Compared with the multi-channel GTX setup in previous measurement, the result shown in Figure 4.16 was measured based on only one channel enable. The FPGA still use 32-bit SERDES mode but with 500MHz reference clock. The core voltage supply is also slightly raised form 1.0V to 1.05V. In the figure we can observe continuous 16Gbps eye with PRBS-31 data pattern, the jitter performance is ~16ps peak to peak and the rise and fall time is about 30ps. Later we tried to raise the data rate to 17Gbps but the eye were closed under this data rate. Therefore we can confirm the speed limit of 28nm Kintex-7 FPGA is around 16Gbps and this data rate just match most up-to-date PCIE 4.0 standard which is coming into the market.

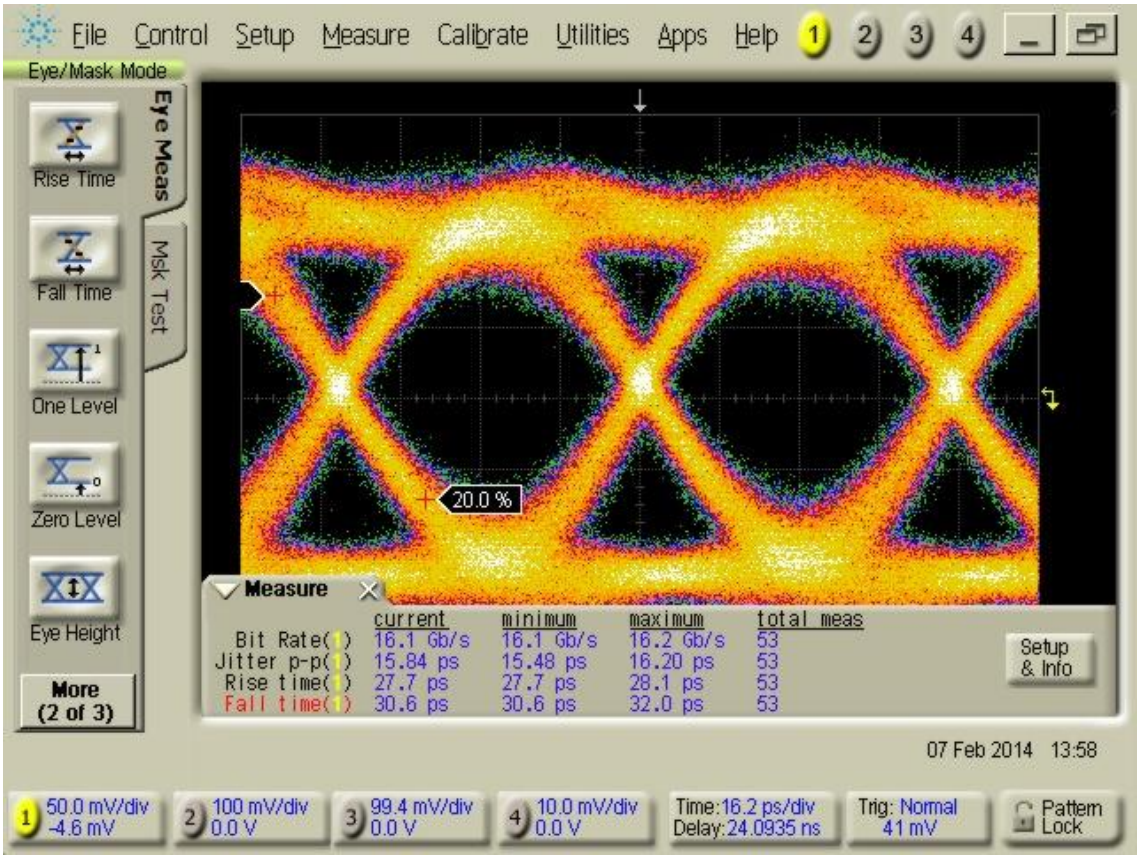


Figure 4.16: 16Gbps GTX signal with PRBS-31 data. The voltage swing is 350mV single-ended. Time base is 16ps/div.

4.4 Summary

FPGA-based Test Platform provides high-performance and high-quality signals up to 10Gbps, which has testability for most of current and coming testing applications. Also the re-configurability and high flexibility of FPGA makes this board a high potential to fit in various testing scenarios. The extension slots on this platform are feasible for designers to build new extension boards and upgrade existing platform for future testing challenges with a relative low cost. This feature also gives an additional benefit of applying this platform to the industry. Furthermore, the power consumption of the testing platform is very low therefore it can get rid of those heavy power supplies and liquid-based cooling system. There is a brilliant future to build a very low-cost and high performance testing system based on this test platform and coming extension boards. More works can be and should be done based on the current work, such as designing new extension testing modules, increasing the number of FPGAs to achieve more testing channels and updating the platform with more advance FPGAs. In short, this architecture has a great opportunity to be developed as an ideal test platform to replace those out-of-dated testers.

CHAPTER 5

EXTENDED APPLICATIONS FOR FPGA-BASED TEST PLATORM

5.1 Introduction

The FPGA-based Test Platform illustrated in the Chapter 4 has provided the possibility of replacing conventional ATE systems [67] [68]. From all aspects including system performance, power consumption and building cost of this platform have shown the competitiveness over ATEs. Despite all these advantages, the most important feature is the flexibility of the platform. The specific-purpose plug-in cards can be realized on the platform to extend the existing functions. In this chapter, we will introduce two new extension boards to show the enhancements on this FPGA-based Test Platform.

The first section will mainly discuss a pin electronic (PE) testing board [58] [65], this board takes up to four GTX channels from FPGA and adds a fine tuning of signal characteristics such as amplitude, pre/post emphasis, phase and DC level adjustment at 3.2Gbps. Simultaneously, this board also offers the ability of detecting the signal amplitude as low as 10mV and the ability of recovering the received signal. In the second part, we will focus on an Ultra-High-Speed (UHS) testing board design [64] [66], this board has extended current 10Gbps+ FPGA GTX signal to 40Gbps. Since FPGA-based Test Platform itself might have limitations at performance and signal quality for future applications, these two boards actually have extended the product life of this platform. With the similar method, more other specific-purpose boards can be developed on this platform and extends the utilization of FPGA to different testing applications.

The experiment results will be well-illustrated in the final section of this chapter. We will integrate the FPGA main board and the two extension boards to show 3.2Gbps pin electronic testing and a very-low amplitude data recovery on PE board, followed by a 40Gbps PRBS-31 signal presented on UHS testing board.

5.2 Pin Electronic Testing Board

5.2.1 Core components evaluation

The main components of the PE board are a programmable Delay chip [69] and a high-performance Driver [70] for transmission side, a latched comparator [71] and a clocked comparators [72] for receiving side, and a resister-splitters with relay to perform bi-direction signal transmission. The control pins of the components on this board can be either controlled by a digital to analog (DAC) chip or external power supply.

The programmable Delay IC on this PE board design is used to adjust the skew of the Driver output signal as well as the delay control of the Shadow Sampler Clock. Micrel SY89297 device was selected because it provides the needed dual-channel delay capability with fine resolution and wide delay range. It is also a relatively small, low-power device to fit this small extension module. The delay value is programed by a SPI interface and the delay range is specified from 5ps to 5ns (5ps step). The SPI interface is realized inside FPGA and will be well-illustrated in later section. This device requires calibration to determine the actual LSB value, from which the 1024 unique delay codes can be estimated. Because of the internal logic structure within this IC, there are 10 independent parameters that are related to the LSB by integer powers of 2. In principle, any of these can be measured and then used to estimate the LSB. We used the largest parameter, representing a delay of 2^9x5ps = 2560ps approximately. The measured delay was then divided by 512 to find the LSB value (near to 5ps). When we calibrated the Driver signal delay path using this approach, we estimated the LSB to be 4.42ps. Then, using this value, we can estimate the other 9 delay parameters. Knowing all 10 parameters, we can accurately predict the actual delay for each of the 1024 delay codes. A plot of the entire range of delay codes, showing some sample codes is provided in Figure 5.1. Here the decimal value of the binary delay code is shown on the X axis, and the delay value (in picoseconds) is shown on the Y axis. The red curve plots the actual measured delay, and the blue curve plots the expected delay in specification. The example of demonstrating the 5ps delay step is shown in Figure

5.2, which the first edge is program to 0ps delay, and the following edges are programmed to 5ps, 10ps and 15ps respectively.

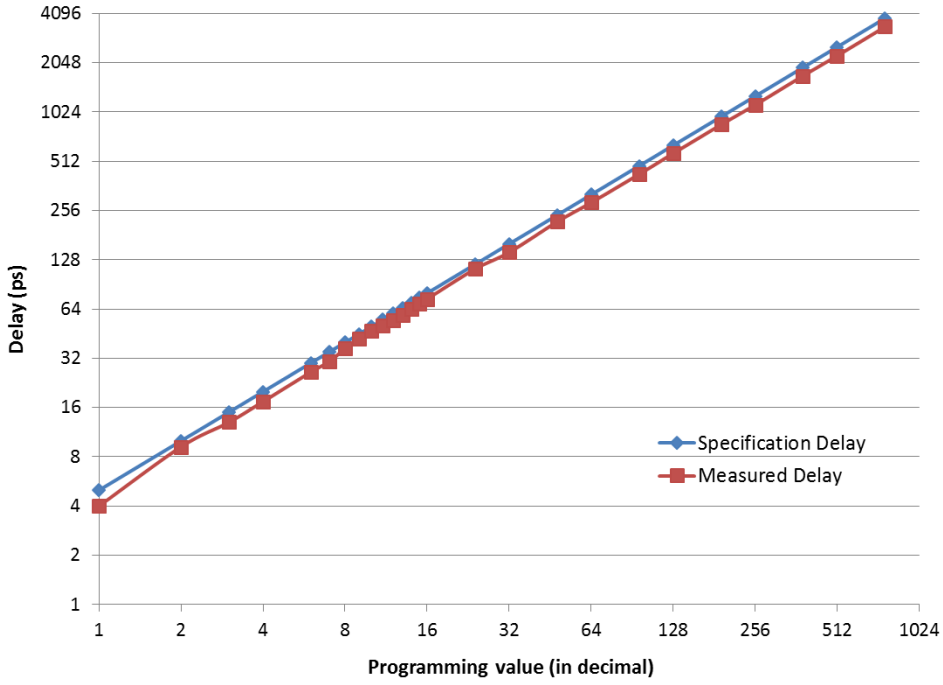


Figure 5.1: Delay linearity verification of the delay chip.

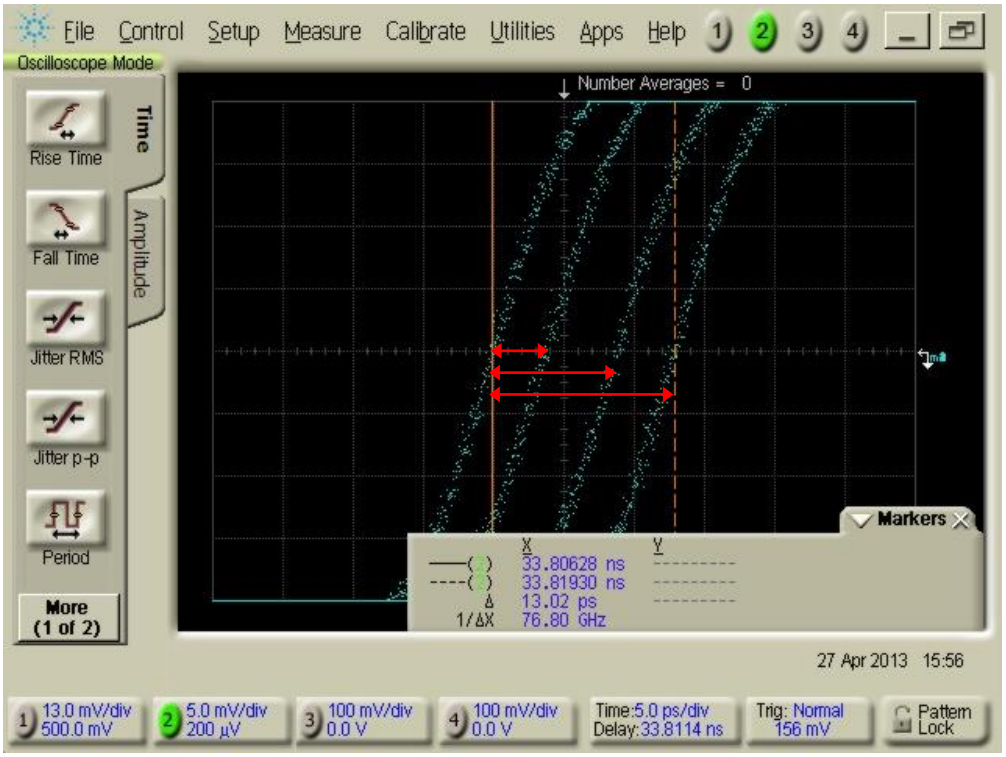


Figure 5.2: Demonstration of ~5ps delay step resolution (5ps/div).

The Driver is the core of the transmission side, which must be able to handle I/O rates of at least 3.2Gbps without significant distortion. It also must have a programmable output swing of ~100mV to ~1.5V and about 1.5V DC offset range, and not add significant jitter (<1ps RMS, <10ps DDJ). In addition to these basic properties, the device must have a small footprint (<1cm^2), and not dissipate too much power (<1Watt) in order to allow for high-density (high pin-count) interfaces on the PE board. Furthermore, the ability of receiving and providing differential output signals is also required. Other features that are desirable include: pre-emphasis control, voltage swing control, DC level adjustment, and reasonable cost. The Micrel SY58626 device was selected since it meets all the requirements and desired features, the block diagram of this IC is shown in Figure 5.3. In the figure we shows a 3-bit control for pre-emphasis magnitude, a 2-bit control for pre-emphasis duration, 2 analog control ports for both voltage swing (TXVCTRL) and DC level ($V_{ref\text{-}ctrl}$). The Driver has two differential channels support while only one (TXIN, /TXIN) is used in PE board. Also both emphasis/driver configuration mode (TXQ, /TXQ) and bypass mode (TXLBQ, /TXLBQ) are included in this IC.

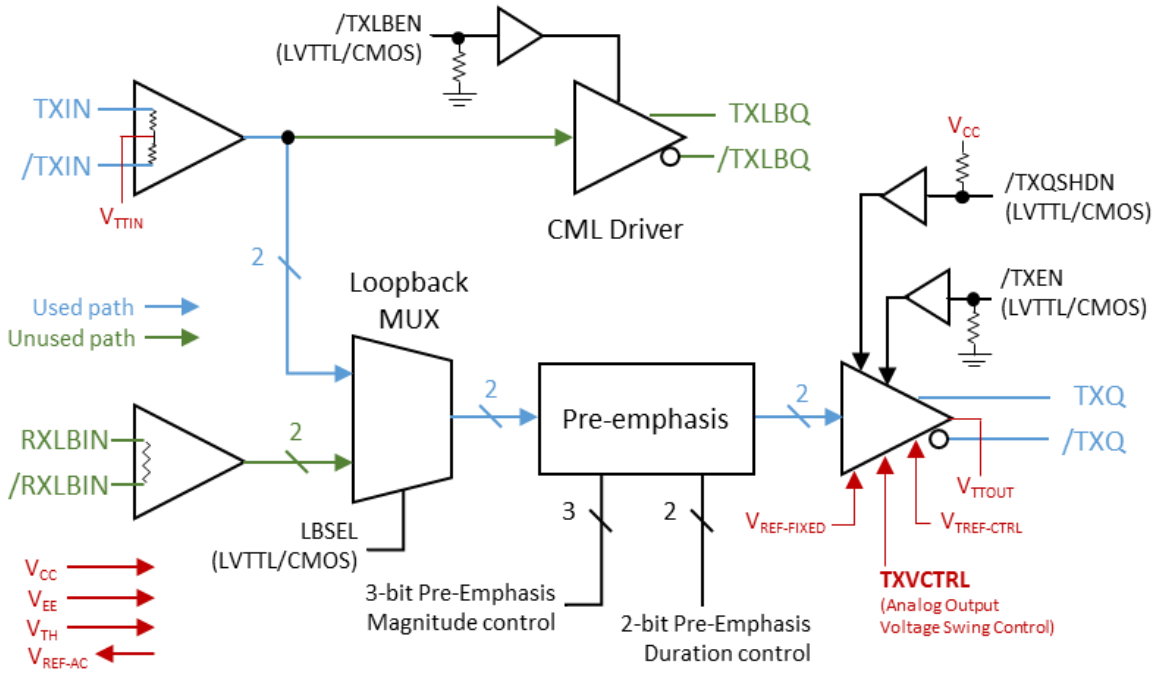


Figure 5.3: The block diagram of Driver chip.

The relationship of pre-emphasis and swing control on the output signal is shown in Figure 5.4. The driver can provide up to 3V peak to peak differential signal and maximum 33% pre-emphasis magnitude and up to 400ps pre-emphasis duration. The performance of this driver will be later shown in experiments section.

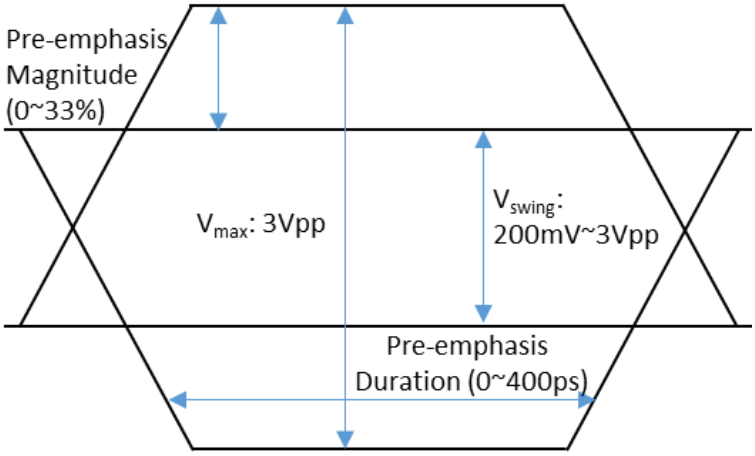


Figure 5.4: Pre-emphasis and voltage swing relationship on output signal.

The RX side is composed of a primary comparator and a shadow comparator. The primary comparator is used on PE board to capture the DUT response, digitize it based upon a user-defined threshold voltage, and to produce full-swing differential output to the FPGA GTX receivers (RX). Therefore this comparator must be able to operate with single-ended input signal and produce fully-differential outputs for the FPGA. It also must operate in "transparent" mode, i.e. without additional clocking. Furthermore, a sensitivity to small (<50mV) single-ended input signals is required, while supporting ~10GHz bandwidth and low jitter with small package(~3x3mm), and dissipate minimal power (<100mW). The Analog Device HMC674 is able to detect as low as 10mVpeak to peak input signal and output 450~1000mV differential sampled signal with 0.2ps RMS jitter and 2ps deterministic jitter. The rise/fall time is as sharp as 20ps and propagation delay (input data to output signal) is about 85ps. The secondary comparator is used for under-sampling the DUT response signal in the form of a "shadow" sampler. It shares all the requirements of the primary comparator, but additionally requires low-jitter clock input(s) for latching

the sampled data bit. Analog Device HMC874 is the clocked version of HMC674 which works well for these requirements. The timing diagram of HMC874 is shown in Figure 5.5, the comparator samples data at the rising edge of the input clock. The example of detecting a 10mV p-p, 3.2Gbps signal with clocked pattern is shown in Figure 5.6.

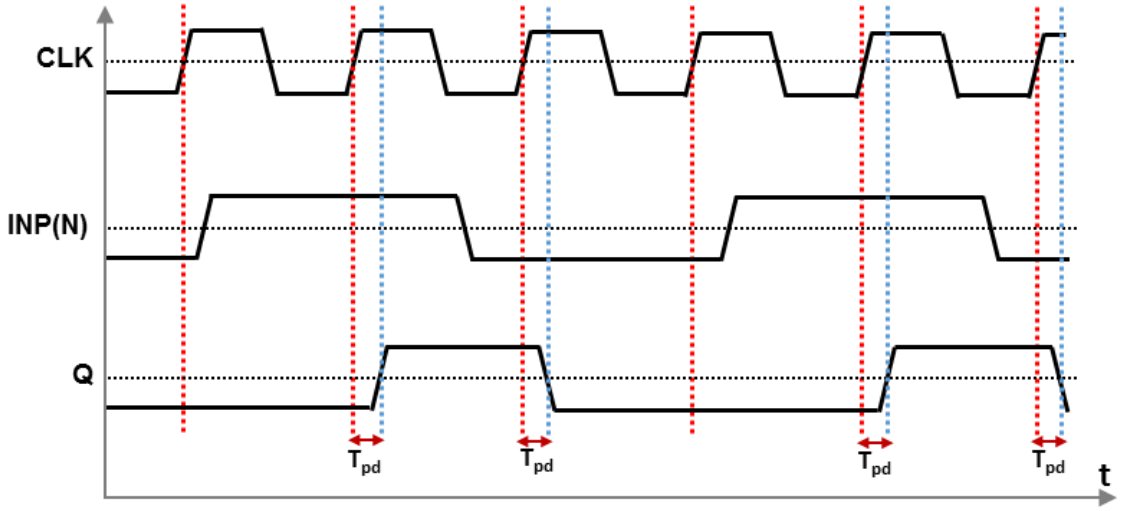


Figure 5.5: The timing diagram of clocked comparator.

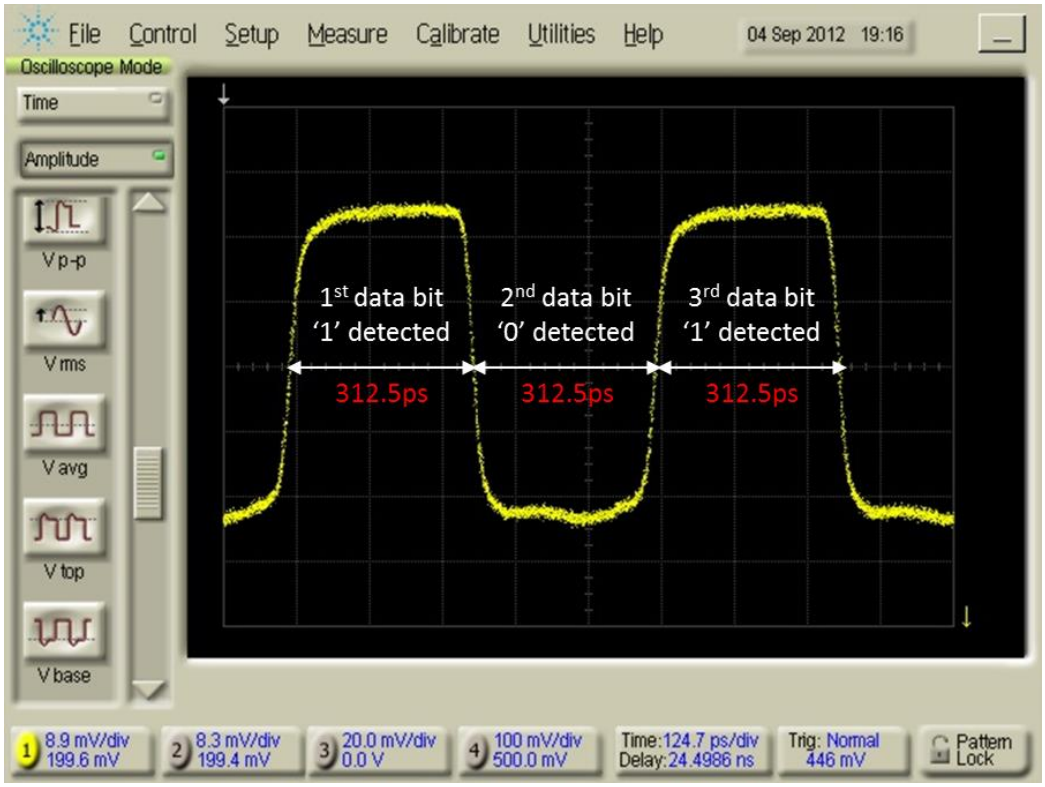


Figure 5.6: The demonstration of for clocked comparator to detect 10mV signal. The input signal is 3.2Gbps and clocked pattern, sampling clock is 3.2GHz.

In addition to the core TX/RX components, additional peripheral components are used to assist the full function of this board. The resister-splitter is used to match 50 ohm impedance of both transmission and receiving side. The single-ended output form driver will be fanout three ways for one transmission and two receivers, therefore the impedance need to be well organized to minimize the signal distortion, the detail information will be discussed in *Hardware Design* section. A high-performance Teledyne relay (GRF312, DC to 8GHz) [73] is implemented for switching between different channels. All the DC pins are control by a Linear Technology 8 outputs DAC (LTC2656) [74], and the swing control pin of the Driver requires high current so that an amplifier LT3080 [75] will be used to magnify the current form DAC outputs. The DAC is also configured by the SPI interface built within the FPGA and will be discussed in later section.

5.2.2 Hardware Design

The PE board is designed for four-channel and bi-directional pin electronic testing. Figure 5.7 illustrates block diagram of one TX/RX channel (total four) on the PE board. The PE board consists of a programmable Delay IC that takes high-speed differential signal from the Xilinx Kintex-7 via a Samtec multi-pin connector. The following Driver chip offers pre-emphasis and amplitude control of the delayed signal. The output signal of the Driver passes through a resistive “splitter” junction that produces three-way outputs. Two of the splitter outputs are “probed” using 200 Ohm series resistors together with 50 Ohm terminations within the two comparator ICs. These 200/50 Ohm resistors form a 5:1 resistive divider with an effective impedance of 250 Ohms, so each comparator receives an attenuated version of the voltage at the splitter junction. Most of the signal energy passes through the two 10-Ohm series resistors, then through the relay and SMP to the DUT. For signals returning from the DUT, the active driver forms part of the termination structure (along with the resistive dividers of the two comparators). The symmetric arrangement of the junction resistors provides a 50-Ohm impedance match looking into the junction either

from the Driver or from the DUT. Therefore, the high-speed logic signals pass through the junction without significant distortions or reflections at both TX/RX direction.

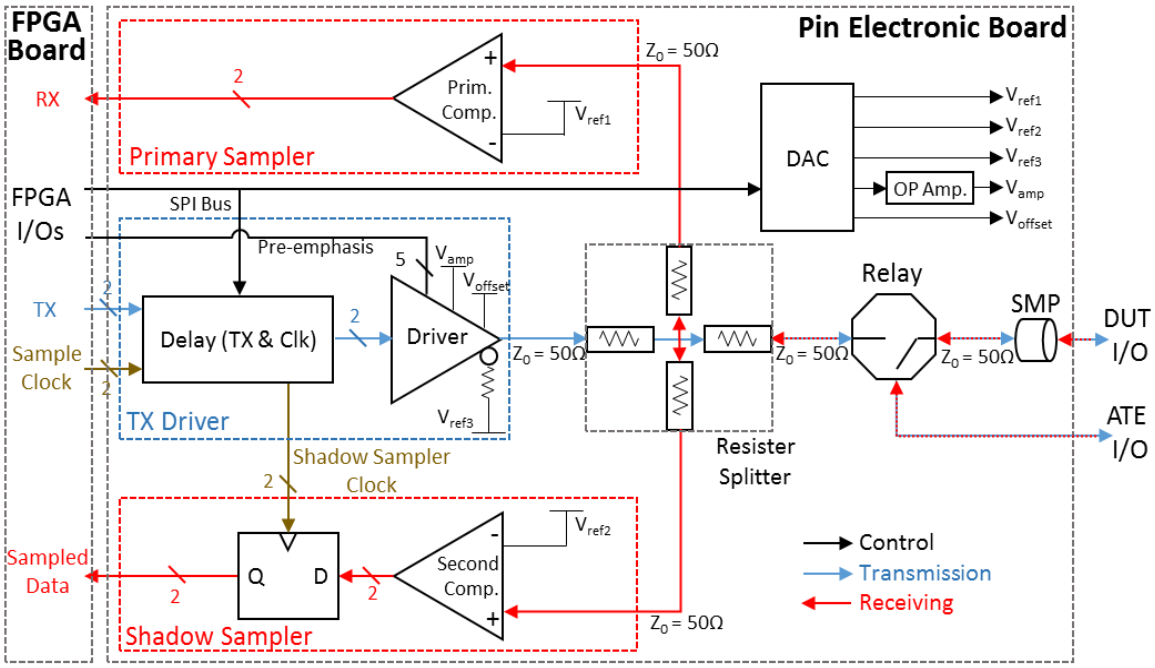


Figure 5.7: Pin Electronics board block diagram.

The Driver compliment output is shown with a terminating load resistor (R = 60ohms), so that the differential current mode logics (CML) circuit remains balanced. In fact we found out experimentally that the CML outputs do not work so well with the termination set to ground. Therefore the design was changed so that the termination was set to V_{ref3}, which is set slightly above the logic "high" level. For this reason, most of our measurements on the PE board were done with an AC-coupling capacitor replacing the first 10-Ohm series resistor, and also AC-coupling the complimentary Driver output signal. Also shown in the figure is an optional 210 Ohm resistor that connects some of the Driver compliment signal to the leg of the junction that produces the signal for the primary comparator. This signal should cancel the attenuated Driver signal for the primary comparator. The only remaining signal component would be that of the DUT. This is an approach called "Simultaneous Bidirectional Signaling" that allows the primary receiver to "see" only the DUT signal, while ignoring the Driver output. Therefore the Driver can

send data even at the same time that the primary comparator is receiving data from the DUT. In this way the PE and the DUT can share the single transmission line for bidirectional communication, while avoid "dead zone" timing conflicts.

The primary receiver is a high-sensitivity comparator, capable of detecting logic signals less than 10mV. It is used to detect and amplify the high-speed signal coming from the DUT, and to create the differential logic signals needed for the Xilinx Kintex-7 GTX receivers. This primary comparator is intended for applications up to 10Gbps. The secondary (shadow) receiver is the other high-sensitivity comparator and capable of detecting logic signals less than 10mV. However, unlike the primary receiver, the secondary receiver has an edge-triggered "clock" input that defines an extremely short sampling window (~5ps). The clock signal is produced in the Kintex-7 FPGA, and is optionally delayed using the Delay IC with 5ps resolution and 5ns range. The secondary receiver is intended to "sample" the signal at the junction and thereby monitor either the Driver output or the DUT output signals. This sampler may gather test information without interrupting the normal functional test sequence.

The main enabler for self-monitoring in our initial design is due to the presence of the "shadow" sampler. Because this measurement element operates independently from the primary sampler, it can be used to gather channel-characterization information even while the normal functional testing is performed (by the primary sampler). Therefore it does not impede the normal functional test process. On the other hand, while the primary sampler normally samples in the middle of the data eye, the shadow sampler can be programmed to observe the signal characteristics near the data-eye boundaries. This is where distortions in the signal will first occur, and is where information about non-ideal operating characteristics is available. In an extreme situation, a failing channel can be quickly identified. However, even slightly-degraded channels may be detected with the shadow sampler. Furthermore, since we are not planning to implement the simultaneous bidirectional signaling approach in the shadow sampler, we will have access to other

information that is hidden from the primary sampler (namely the Driver signal). So we can use the shadow sampler to observe the Driver signal quality and the DUT signal quality, as well as distortions from defective channel transmission lines. In an off-line mode, the shadow sampler will facilitate TDR measurements, Driver/Comparator de-skew and voltage calibration, and channel length calibration. Therefore, the shadow is very effective at enabling many channel characterization techniques.

The five analog control pins are provided by the LT2656 DAC (top-right corner of Figure 5.7). The V_{ref1} and V_{ref2} provide the reference voltage for both two comparator, and the V_{ref3} offers the optional termination voltage of Drivers (since we found the best termination voltage is not ground for the driver). V_{amp} and V_{offset} provide amplitude and DC level control for the driver respectively. Since V_{amp} draws a relatively large current which cannot be generated by the DAC, we added an OP-Amp to amplify the current of V_{amp} and enable the function of amplitude control.

5.2.3 Board Stack up and Layout

The PE board is built of a 10-layer PCB with 2 outer Rogers-4350 Pre layers and 7 inner FR4 layers, and the target thickness is 62.5mils to fit the edge mounted connector. The stack up of the PE board is shown in Figure 5.8. The thickness of 1 OZ Cu is about 1.35 mils, and the Pre-Preg layers are either 4 mils or 8mils (depends on against plane or signal layer). The top and bottom layer (Signal1 and Signal2) are high-speed signal layers since these two layers are built of low-loss Rogers material, the high-speed traces (TX, RX and clock) should be routed on these two layers to achieve best performance. The transmission line on both two layers is Microstrip line, the high-speed traces are designed to be 8 mils width based on the stack up and dielectric of the material to match 50 Ohms impedance and reduce the reflection and distortion. The two power layers are splitter power layers, multiple power plans are realized in each layer. We had built positive power planes (1.5V, 3.3V and 5V) on Power1 layer and negative power planes (-1.8V, -3.0V) on Power2

layer. All the low-speed control traces (analog pins, DC I/Os and SPI interface) are routed on both Inner1 and Inner 2 layer. The composite-layer layout of the PE board is shown in Figure 5.9, which the detail layout of each layer is attached in Appendix B.

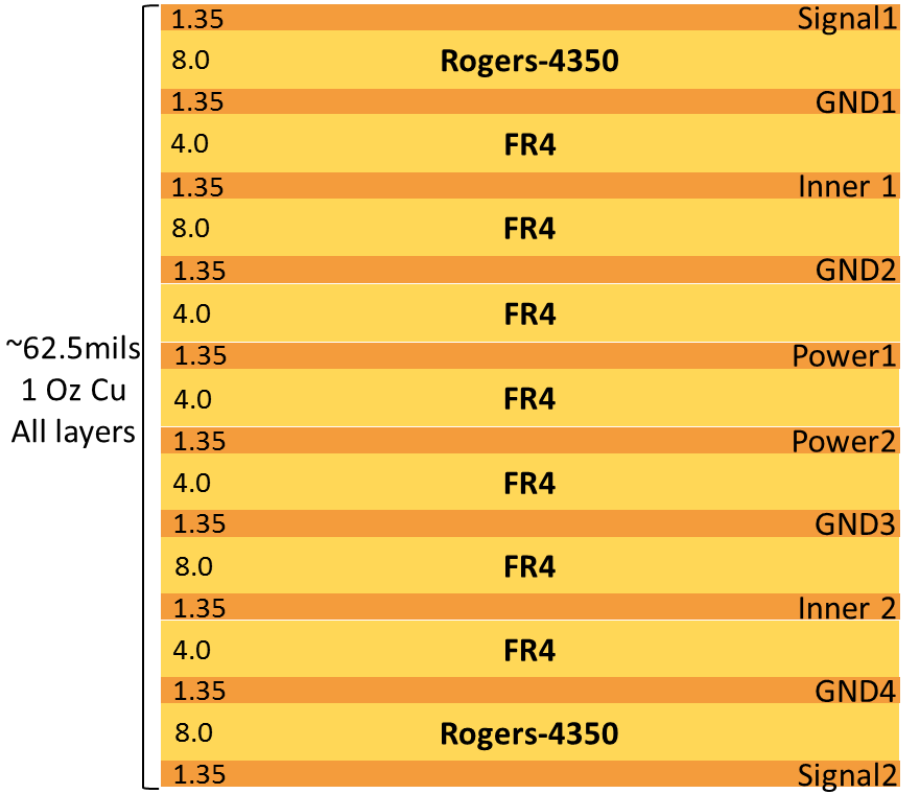


Figure 5.8: The stack up of 10-layer PE board.

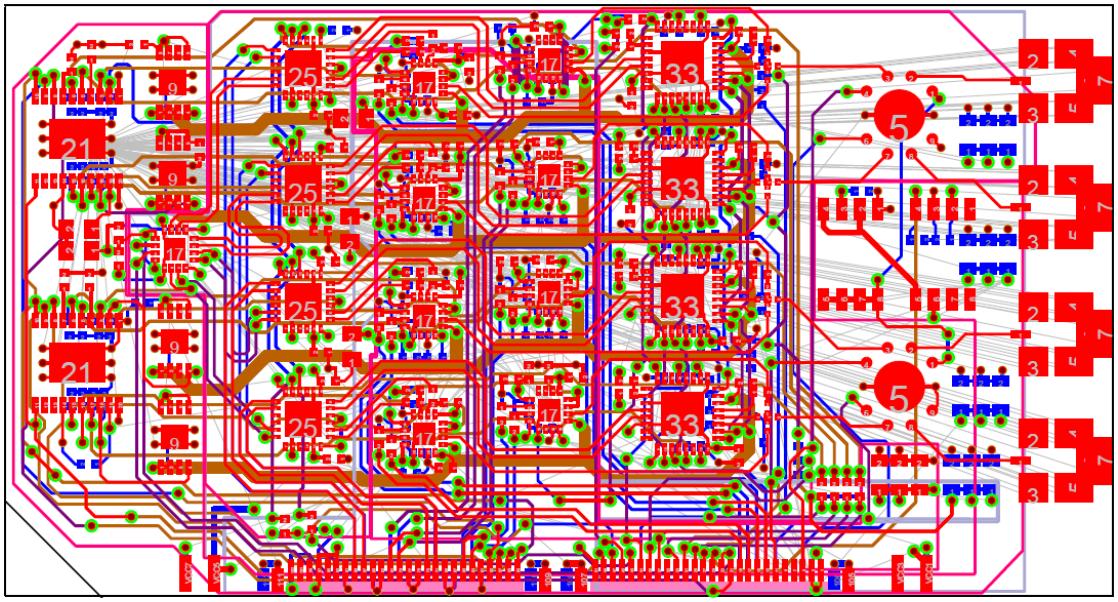


Figure 5.9: The layout of PE board.

5.2.4 SPI-Master controller Design

The delay chip and DAC relay on SPI interface to get programed. The SPI bus is a synchronous serial communication interface specification used for short distance communication, primarily in embedded systems [76]. Typical applications include Secure Digital (SD) cards and liquid crystal displays (LCD). SPI devices communicate in full duplex mode using a master-slave architecture with a single master, as shown in Figure 5.10. The master device originates the frame for reading and writing. Multiple slave devices are supported through selection with individual slave select (SS) lines. Sometimes SPI is called a four-wire serial bus, contrasting with three-, two-, and one-wire serial buses. The SPI may be accurately described as a synchronous serial interface, but it is different from the Synchronous Serial Interface (SSI) protocol, which is also a four-wire synchronous serial communication protocol, but employs differential signaling and provides only a single simplex communication channel.

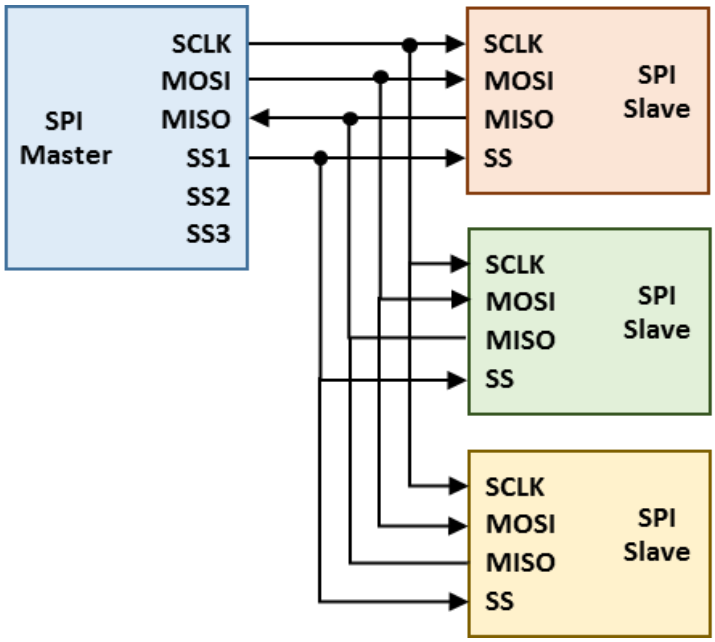


Figure 5.10: SPI bus with single master and several slaves.

Since each SPI-communication device includes SPI-Slave controller already, a SPI-Master controller is required to control those devices. The FPGA is a suitable device in our system to implement SPI-Master controller. The most obvious issue to realize this interface

on FPGA is the frequency of reference clock. FPGAs typically run at several-hundred MHz, however SPI clock usually operate at <100MHz. Therefore a frequency-divider is implemented to divide the reference clock down to appropriate frequency. Although the SPI interface is able to run a master controller with several slave controllers, the operating frequency supports and the programming word lengths are a bit different between the two devices. In order to simplify the controller design, we decided to generate two SPI master controllers to control the two devices respectively.

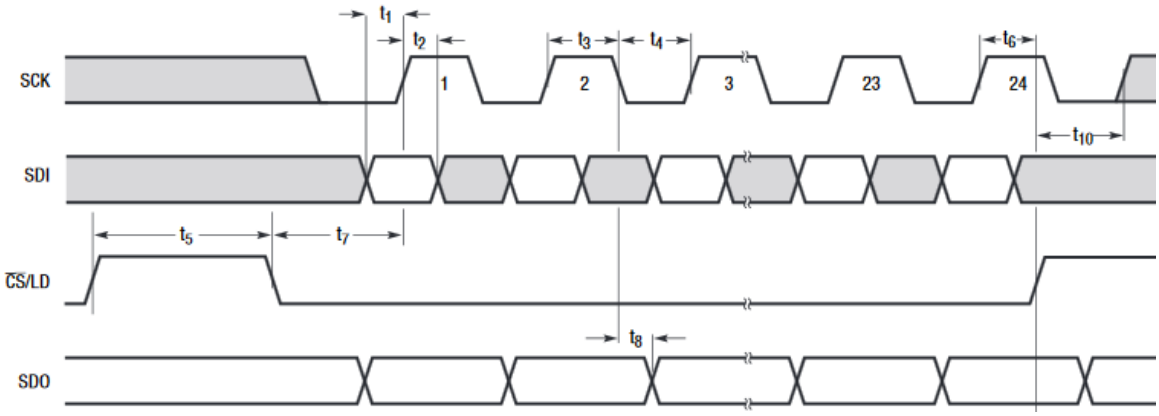


Figure 5.11: The SPI timing chart of DAC [74].

A finite state machine with the divided clock is built to realize SPI timing chart shown in Figure 5.11. First of all we use a "divided-by-10" clock from the FPGA reference clock and divider circuit as SPI reference clock to match the spec of <50MHz). In the initial state, CS/LD pin remains high to prevent data writing. Then the CS/LD goes low and the SCK clock starts toggling, in the meantime the data pipes into (FIFO) the SDI pin with SCK. The SCK is used to sample these data on the Slave side. The programming bit-width for LT2656 is 24 bits, therefore the SCK stops toggling after the 24th cycle. Afterward the CS/LD bit will go back to high to end the write operation. The read operation follows the similar process, the only difference the data read form SDO pin as the SCK toggling. The SPI interface in delay chip is very similar to DAC, some minor differences are the delay chip has different programming bit and slower SCK speed supported. Therefore we will not repeat the SPI interface implementation of delay chip again.

5.3 Ultra-High-Speed Testing Board

5.3.1 Core Components evaluation

The core of TX side on this board is realized by a high-speed Analog Device HMC847 [77] MUX chip. The most critical qualification of TX side is able to receive 10Gbps signal from the FPGA and serialize to even higher speed at output end. The HMC847 is a 4:1 multiplexer (MUX) for operation at output data rate up to 40Gbps. The mux latches four differential inputs on falling edge of the input clock. The device uses both rising and falling edges (DDR) of the half-rate clock to serialize the data. A quarter-rate clock output, which is synchronous to the data output of the MUX, is generated on chip. This clock is used to trigger the 86100D scope later in the measurement section. All clock inputs and data I/Os of the MUX are CML and terminated on-chip with 50 Ohms to the Vcc (3.3V), and can be DC or AC coupled. The I/Os of the MUX can be operated either differentially or single-ended. The MUX chip also has an output level control pin, V_{CTRL}, which allows for signal-loss compensation and signal-quality optimization. The relationship of output voltage V_{CTRL} and is shown in Figure 5.12. Another voltage control pin V_{DCC} controls the data output cross-point & duty cycle. The rise/fall time is both specified about 12ps and the RMS jitter under room temperature is about 0.75ps.

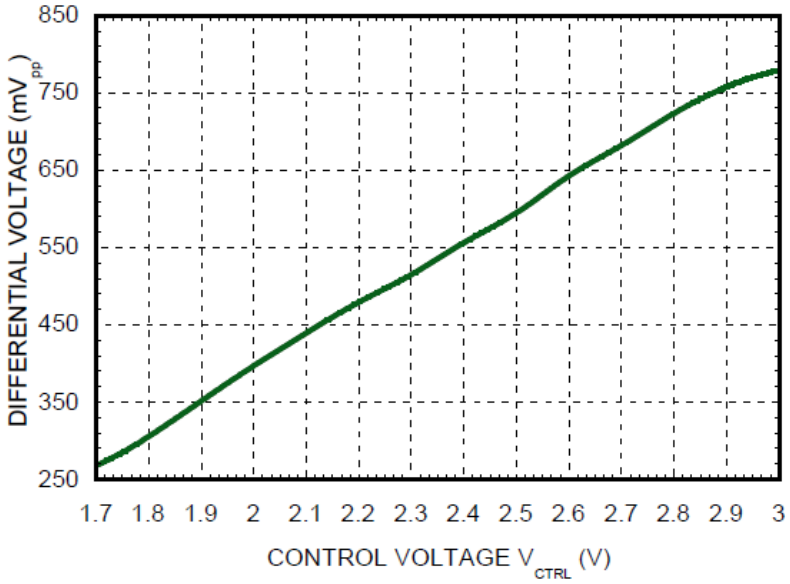


Figure 5.12: Differential output swing vs. V_{CTRL} of MUX chip

In order to achieve the high-speed serializing, we need to have a precise skew-management on the sampling clock. A high-performance delay chip is required to realize this function. The Analog Device HMC911 is a broadband time delay with 0 to 70ps continuously adjustable delay range. The delay control is linearly monotonic with respect to the differential delay control voltage and the control input has a modulation bandwidth of 1.6GHz. Figure 5.13(a) has shown the relationship between delay time and the differential control voltage. The device provides a differential output voltage with constant amplitude for single-ended or differential input voltages above the input sensitivity level, while the output voltage swing may be adjusted using the V_{AC} control pin. The relationship of output swing and the V_{AC} voltage are shown in Figure 5.13(b).

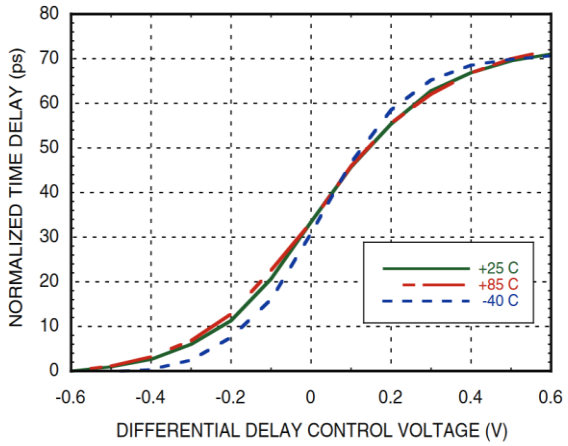


Figure 5.13(a): Time Delay vs Delay Control Voltage of MUX chip

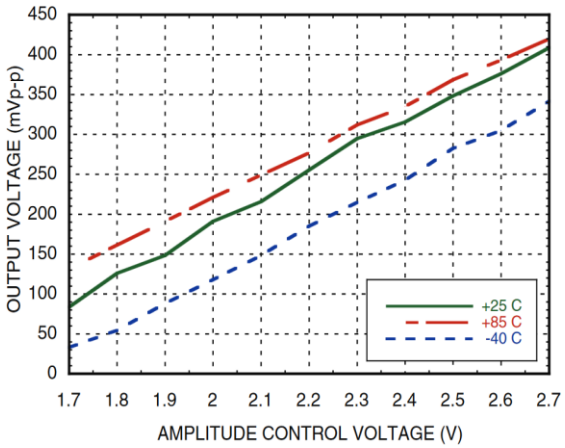


Figure 5.13(b): Single-ended Output Swing vs V_{AC} of MUX chip

To receive this high-speed signal, we use an Analog Device HMC848 [78] to realize RX end. The HMC848 is a 1:4 de-multiplexer (DeMUX) designed for data deserialization up to 45Gbps. Very similar to HMC847, this device also uses both rising and falling edges of the half-rate clock to sample the input data in sequence (D0-D3) and latches the data onto the differential outputs. Also a quarter-rate clock output generated on-chip can be used to synchronize with other devices. All clock inputs and data I/Os of the DeMUX chip are also CML and terminated on-chip with 50 Ohms to the V_{CC}, and can be either DC or AC coupled. The I/Os of the HMC848 can be operated either differentially or

single-ended as the HMC847 MUX. The rise and fall time of this DeMUX output is about 25ps and the RMS jitter is ~0.35ps.

There are several voltage control bits ($V_{AC,}$ V_{CTRL} and V_{DCC}) on all these three chips. Similar to the PE board, we use a LT2656 DAC to control these analog ports, therefore a SPI interface is also needed to be implemented in the FPGA to control this board as well. Also the direct control from power supplies are reserved for debug.

5.3.2 Hardware Design

The FPGA Test Platform described in Chapter 4 has potential to run at even higher speed based on developing an additional plug-in board to improve the bottleneck of the FPGA performance (max at 10Gbps). The basic concept is to build an extension module for another stage of serializing and plugin on the existing FPGA test platform. The simplified operation can be addressed as follows: In transmitter side, the extension board takes four 10Gbps high-speed signals from the FPGA platform, and MUXs those signals to serialize a 40Gbps signal. On the receiver side, looping this 40Gbps signal back to a high-performance receiver, the receiver de-serializes this signal to four 10Gbps bit-streams so that the FPGA RX is able to receive and recover the data.

Figure 5.14 shows the top-level architecture of this extension board. The power supplies provide voltage through the power plans of platform and pass via high-bandwidth connector to this UHS board. The TX/RX signal and SPI path also pass through this connector. In the figure, the blue arrows show the signal direction of transmission. On the FPGA platform, the GTX transceiver generates 10Gbps signal with arbitrary pattern. Then the high-speed connector brings four-channel differential 10Gbps GTX signals from FPGA platform to this module. The MUX chip uses a high-speed phase-adjusted reference clock (up to 20GHz, depends on the final data rate) from one end of the delay chip differential output to serialize these GTX signals to form an up to 40Gbps signal. The delay chip takes the reference clock from an E8257D signal generator and has a maximum 70ps delay

control ability. This delay chip provides the ability of controlling the sampling phase of the serializer, which is critical for synthesizing the high-speed signal.

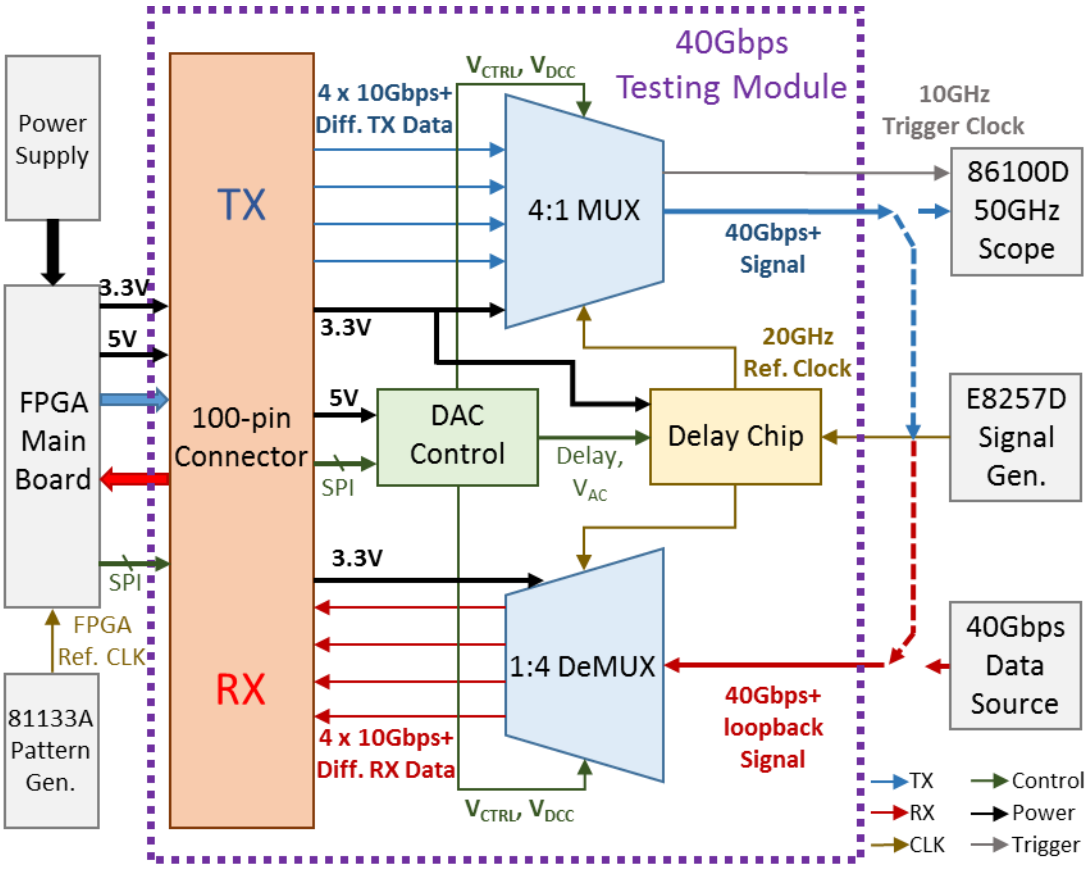


Figure 5.14: UHS extension board block diagram.

The detailed timing diagram of the MUX is illustrated in Figure 5.15. The four sampling edges (two rising edges and two falling edge) of the high-speed reference clock must be aligned within the eyes of each data pin (D1~D4) to get best sampling. If any of these sampling edges is placed at the time of data transition, then we will get erroneous sampling. For example, if we take 10Gbps signals from the FPGA, there is only a 100ps period time slot available for sampling four times (two rising edges and two falling edges). Although we have optimized the skew of each data channels and the sampling clock by tuning the trace length, there might still exist several picosecond differences between the data channels and the reference clock due to the PCB manufacturing variation and different PVT conditions. Therefore, a more reliable adjustment is required for the user to find the

optimal sampling phase. The 70ps variable delay from the delay chip provides about 0.7UI (for 10Gbps data) adjustment to optimize the sampling location, this delay range should be enough for us to find the appropriate sampling location. In the meantime, this serializer will also generate a divided-by-two clock, which is 10GHz in the example. Since we mainly use a sampling scope (Keysight 86100D) to do our measurements, a low-jitter triggering clock is strongly required to get the best measurements. Therefore the clean output clock from this serializer is a decent source to trigger the 86100D scope and measure the final ultra-high-speed signals.

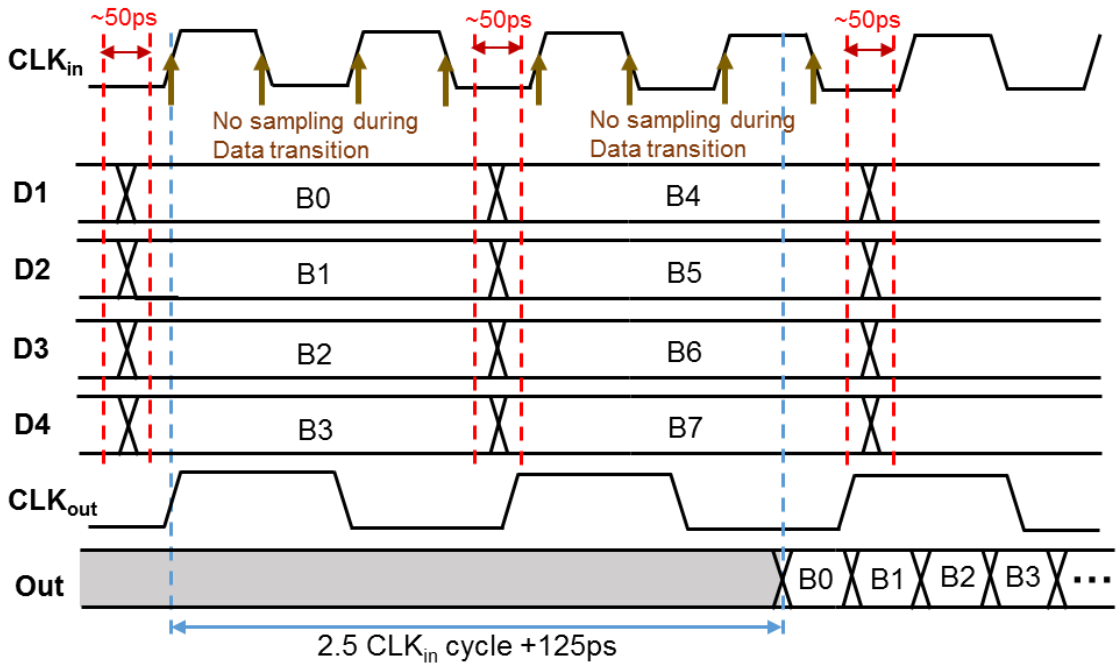


Figure 5.15: Timing diagram of multiplexing process.

This board is designed not only to generate but also to capture 40Gbps signals as well. Since we don't have a PRBS generator that is able to generate such high data rate signal, the solution is to loop the signal from the transmission side back to receiver side to verify the received data. The red arrows in Figure 5.14 shows the direction of signal receiving path. Since the FPGA is only able to transmit/receive 10Gbps signals, it is necessary to perform pre-deserializing process of the 40Gbps signal before received by the FPGA. The receiving operation can be addressed as follows: The loopback 40Gbps signal

is first received by an HMC848 DeMUX chip. The de-serializing process is very similar to serializing but work in opposite direction. This chip takes the other end of the delayed reference clock from HMC910 and applies this clock to recover the 40Gbps signal into four-channel 10Gbps signals. Then four de-serialized signals pass through Samtec connectors and received by the FPGA GTX receiver. The FPGA receivers recover the data and use built-in PRBS pattern checkers to check the receiving data pattern.

The control pins on delay chips, MUX and DeMUX parts can be controlled either by a multi-channel LT2656 DAC, trimmer-pot, or directly from power supplies. The SPI –Master controller for DAC control is implemented on FPGA main board. The interface between this extension board and FPGA board is a 10GHz-bandwidth Samtec connector which is used to carry ~10Gbps GTX signals, power supply and low-speed I/Os. All the connectors used to carry very high frequency (>10GHz) signal are either high-performance SMA or SMP connectors (>26.5GHz bandwidth). Since the board is running at extremely high-speed, appropriate decoupling capacitors are also soldered to eliminate the high-speed noise added on the power supply.

5.3.3 Board Stack up and Layout

The UHS board is built of a 4-layer PCB with FR4 material and the target thickness is 62.5mils to fit the edge mounted high-bandwidth connector. Due to the relative high cost of those MUX/DeMUX chip and delay chip, we decided to use more economical way to build this board. The stack up of the UHS board is shown in Figure 5.16. The thickness of 1 OZ Cu is about 1.35 mils, the inner core is 40 mils and the outer Pre-Preg layers are about 8 mils. Therefore we can build a ~62.5 (+/-10%) mils thickness board. The top and bottom layer (Signal1 and Signal2) are high-speed signal layers, the high-speed traces (TX, RX and reference clock) should be routed on these two layers. Those transmission line on those two layers are Microstrip line, so that the high-speed traces are designed to be 12 mils width based on the stack up and dielectric of the material to match 50 Ohms impedance

and reduce the reflection and distortion. The only power layer is a splitter power layer, therefore both 3.3V (MUX/DeMUX and Delay chip) and 5V (DAC chip) power plans are implemented on this layer. All the low-speed control traces (analog pins, DC I/Os and SPI interface) are also routed on both Signal1 and Singal2 layer but with lower priority. Furthermore, a lot board space on both two signal layers is reserved for the decoupling capacitors. The composite-layer layout of the PE board is shown in Figure 5.17, which the detail layout of each layer is attached in Appendix B.

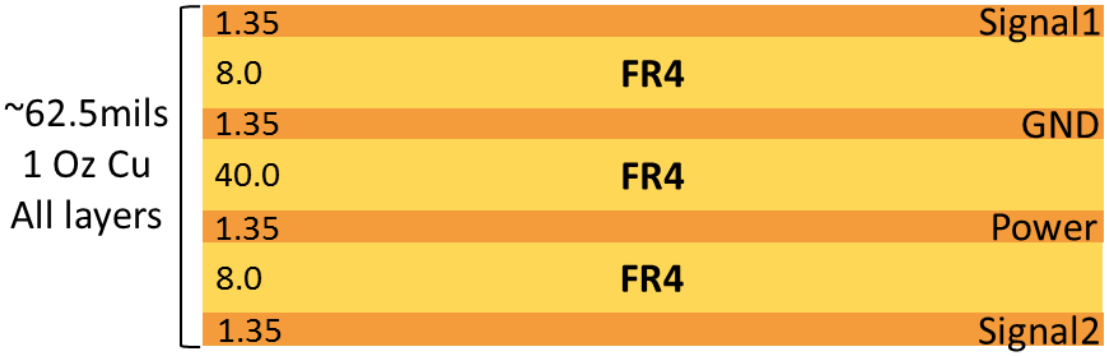


Figure 5.16: The stack up of UHS extension board.

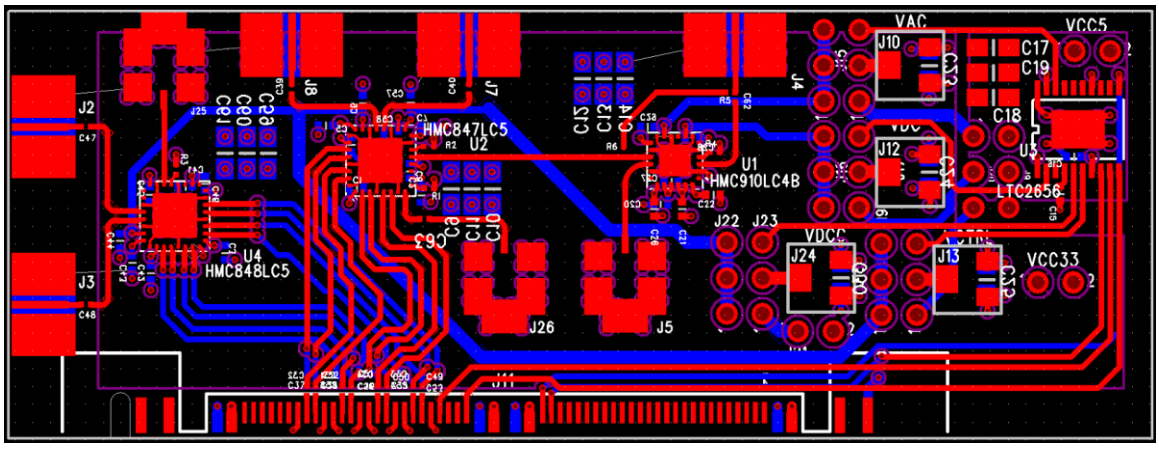


Figure 5.17: UHS extension board layout.

5.4 Experiments and Testing Results

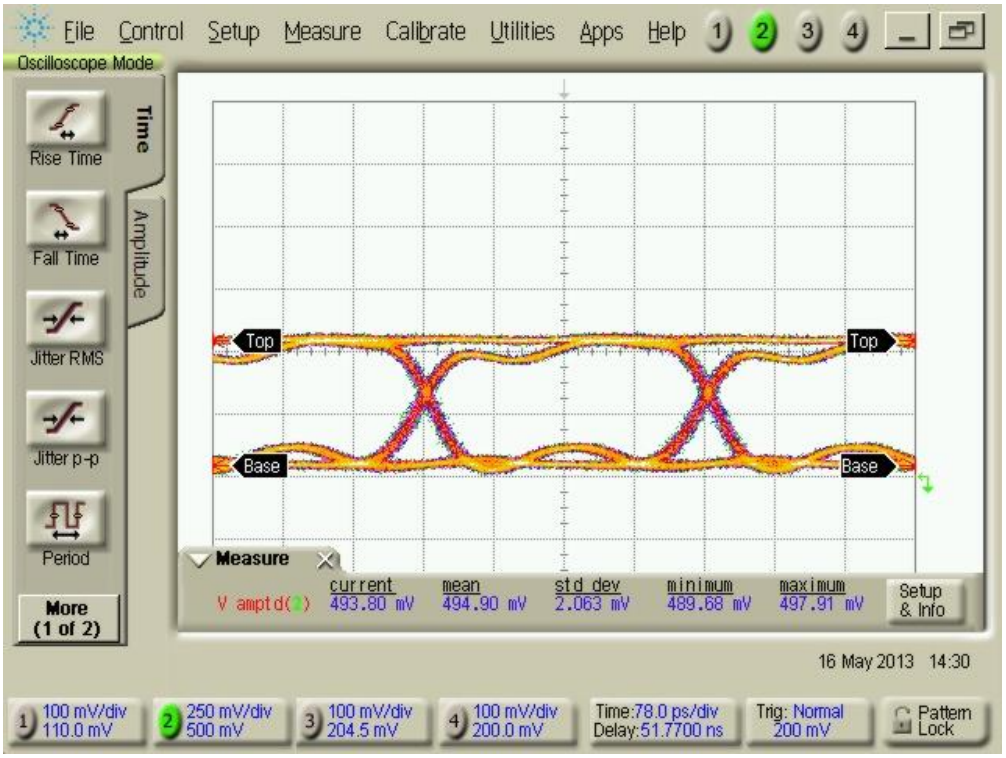
5.4.1 Pin Electronic Testing Board

The FPGA-based Test Platform serves as a central controller for the high-speed signals provided to, and obtained from the PE board. The general configuration is programmed on the FPGA main board. In this section we will test the signal's performance and pin electronic characteristic from PE board. The signal data rate transmits/received on PE card will be fixed at 3.2Gbps and we will try different settings to characterize pin electronics (pre-emphasis, voltage amplitude and DC offset) and demonstrate the flexible-testing feature coming out of the PE board. Also the performance of the low-amplitude receiver will be shown in following experiment. In the end we will push the PE board to run at 5Gbps with appropriate setup of emphasis.

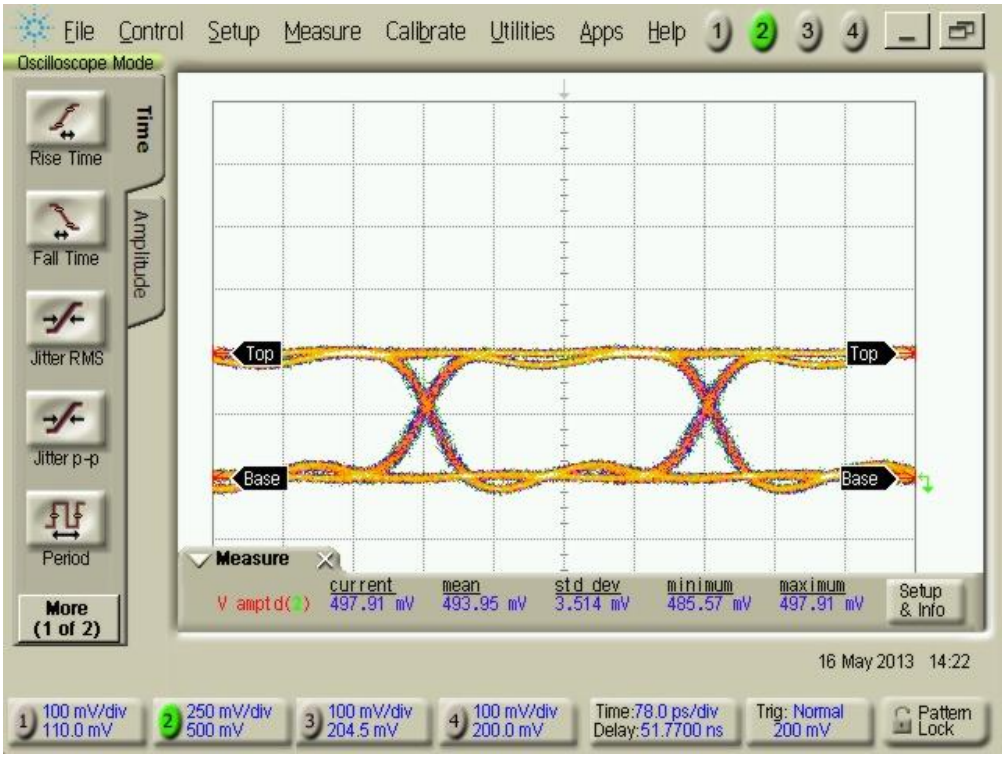
5.4.1.1 Pre-emphasis duration and magnitude characterization

The Driver chip has five digital programming bits for adjusting the voltage-amplitude and time-duration of the pre-emphasis added to the nominal signal waveform. Three bits are programmed from 0% (no emphasis), 10%, 15%, 25%, or 33% for pre-emphasis magnitude, and the other two bits are used to set the pre-emphasis time-duration as 60ps, 100ps, 200ps, or 400ps (only at the data rate <3.2Gbps).

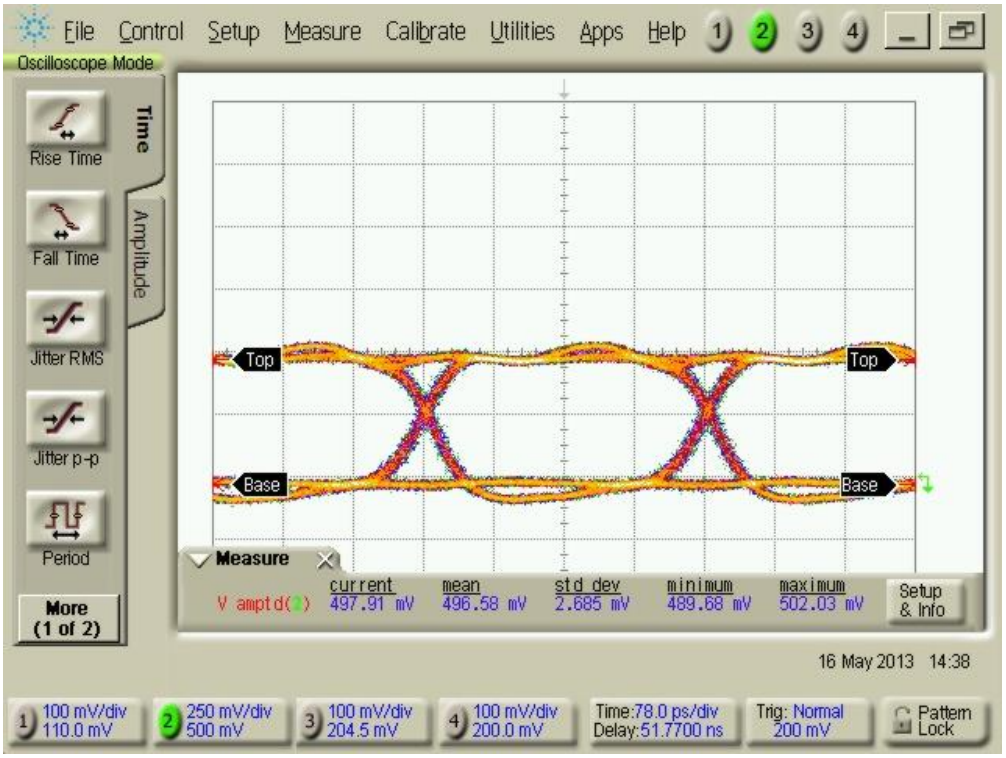
First of all, we fixed DC offset at 250mV and signal amplitude to 500mV, and tried several pre-emphasis setup combinations (both magnitude and duration) on driver chip to achieve the best-quality waveform at 3.2Gbps data rate. Figure 5.18 shows and compares signal quality under fixed pre-emphasis magnitude with sweeping pre-emphasis duration. In Figure 5.19, we fixed the pre-emphasis duration to 100ps and sweep the magnitude, therefore we can present over emphasis settings on the waveform. In this experiment, we found that 10% or 15% was the optimal emphasis magnitude level, and 100ps or 200ps was the optimal emphasis duration for the frequencies of interest (1 to 5Gbps).



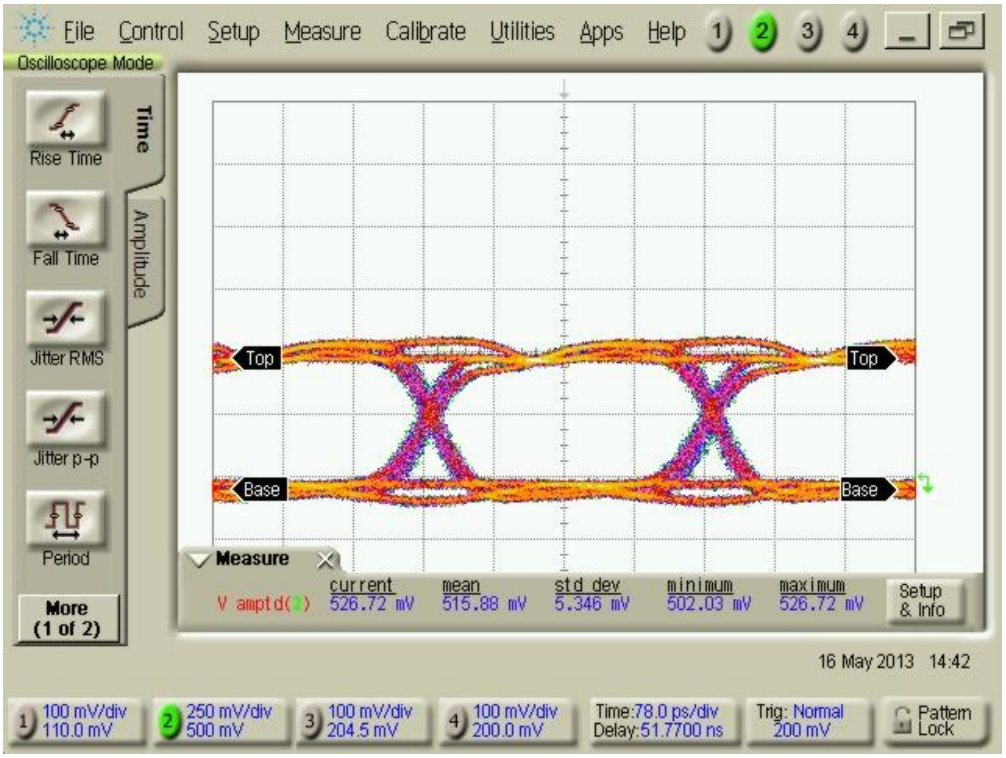
(a) No pre-emphasis.



(b) 10% pre-emphasis magnitude and 100ps pre-emphasis duration.

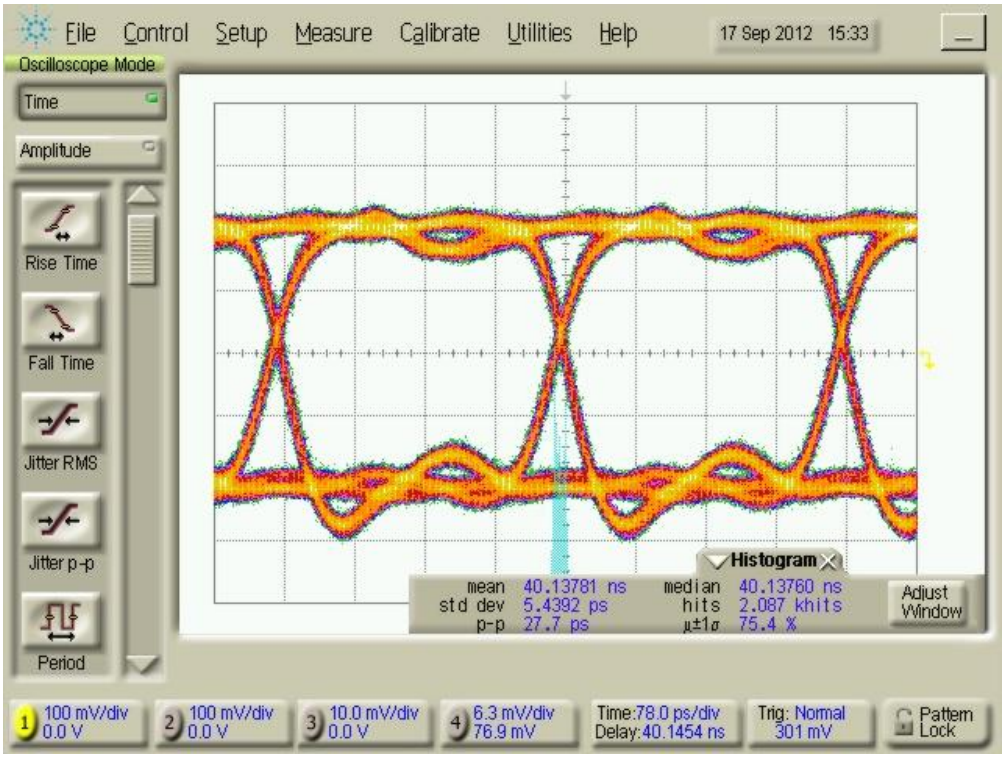


(c) 10% pre-emphasis magnitude and 200ps pre-emphasis duration.

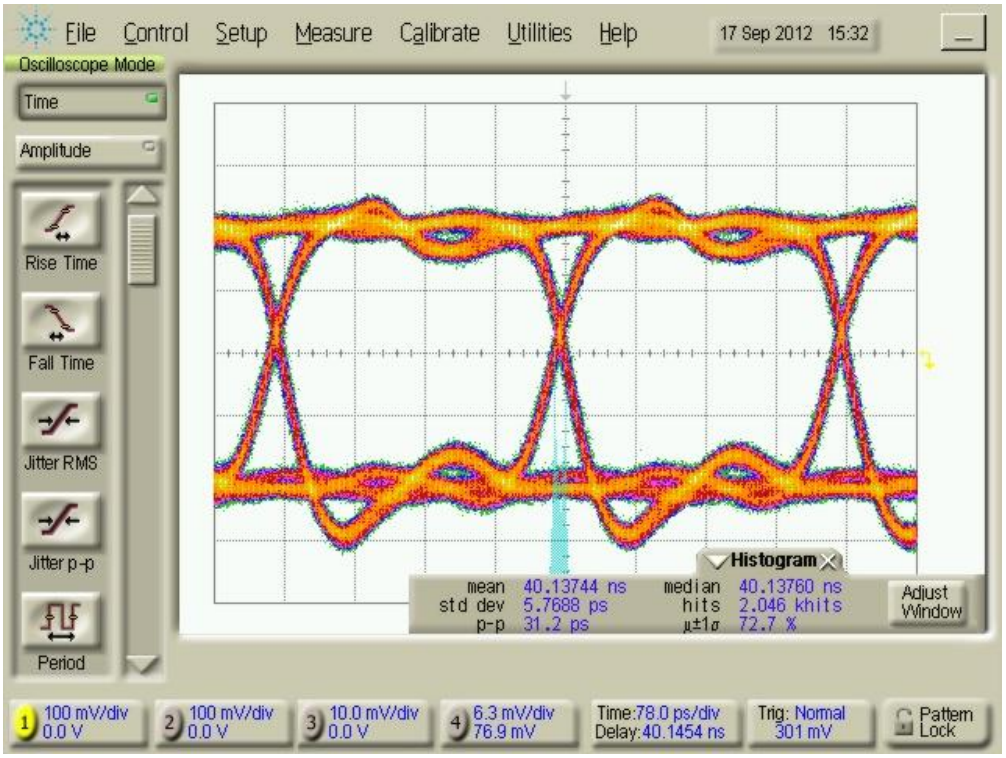


(d) 10% pre-emphasis magnitude and 400ps pre-emphasis duration.

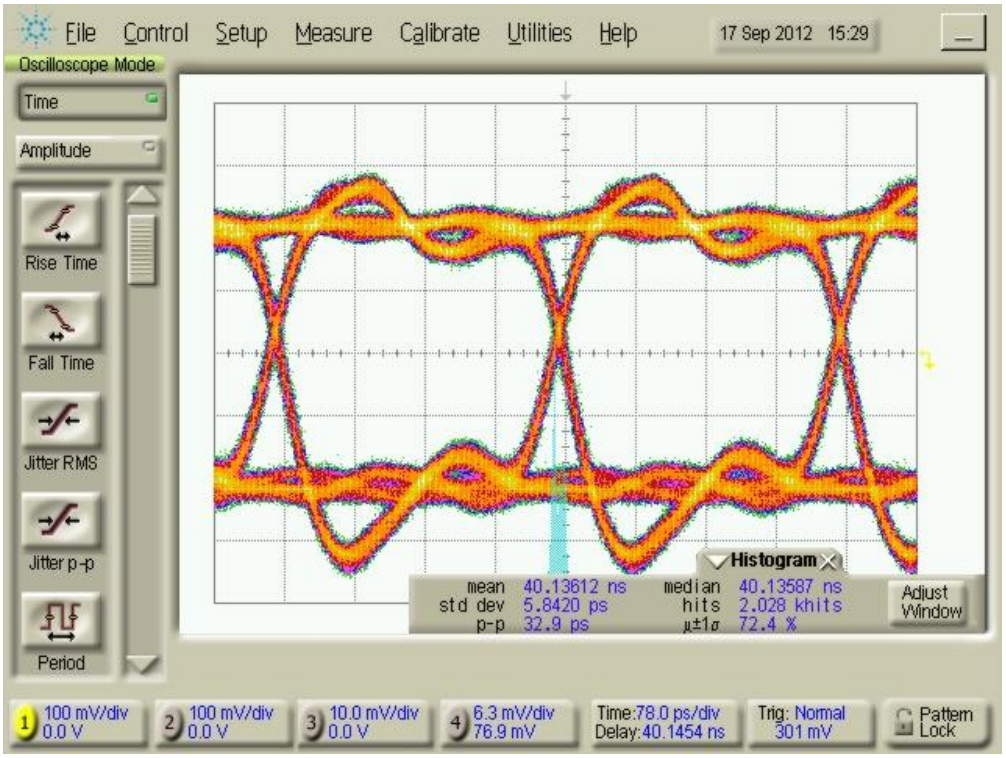
Figure 5.18: Pre-emphasis duration sweeping on 3.2Gbps and PRBS-31 signal.



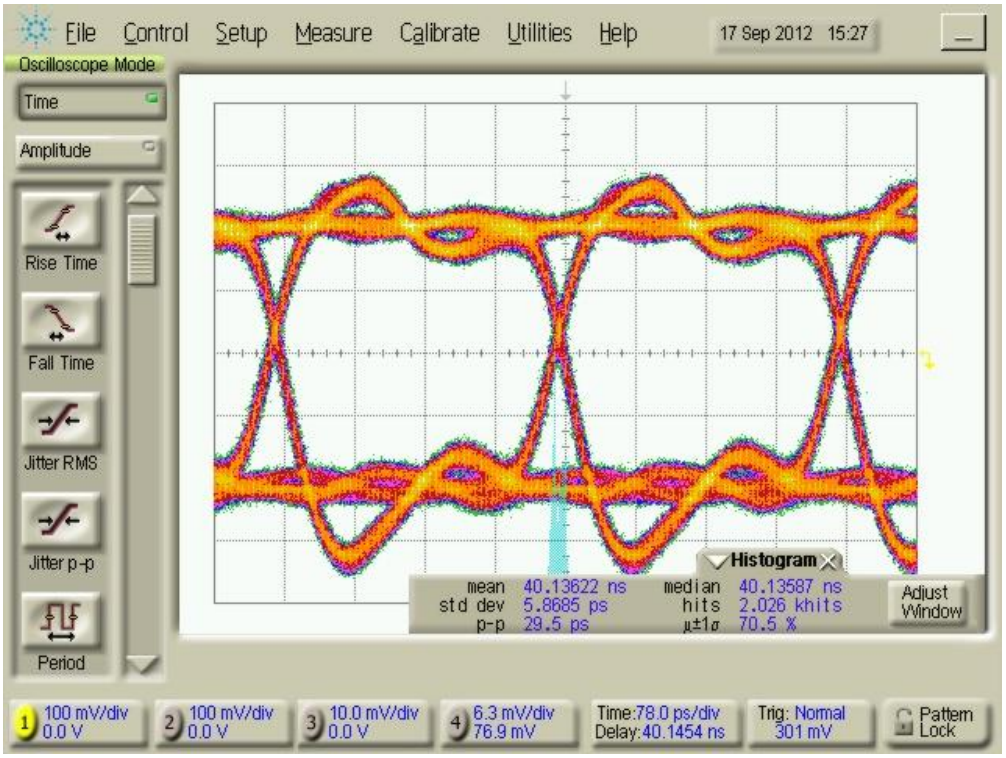
(a) 10% pre-emphasis magnitude and 100ps pre-emphasis duration.



(b) 15% pre-emphasis magnitude and 100ps pre-emphasis duration.



(c) 25% pre-emphasis magnitude and 100ps pre-emphasis duration.



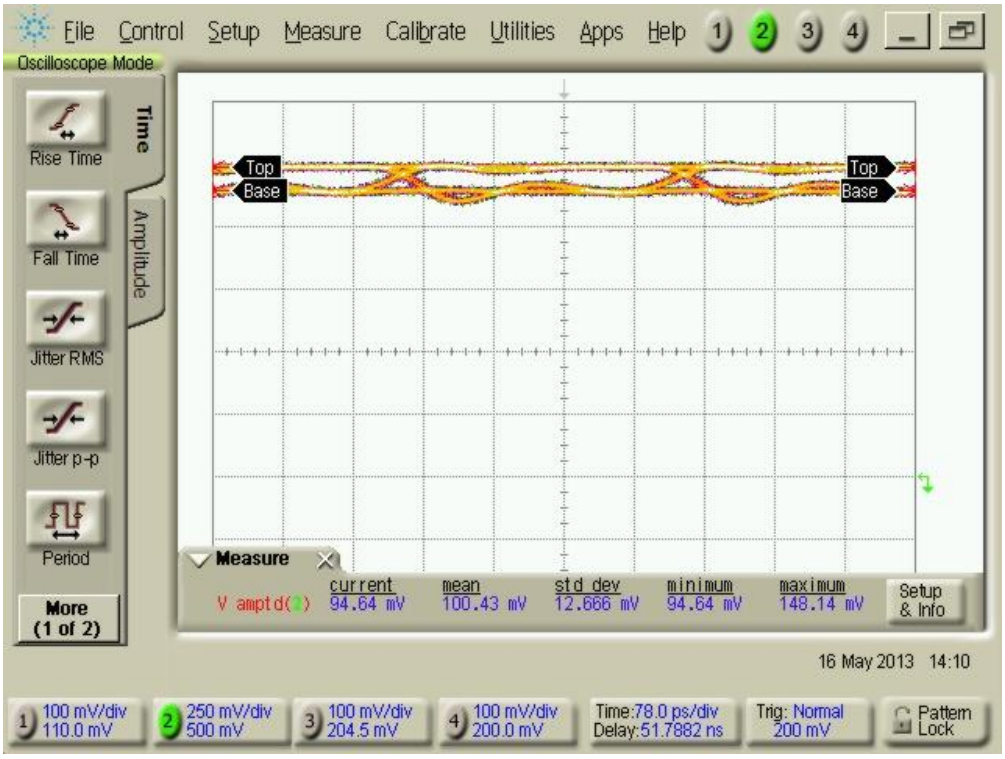
(d) 33% pre-emphasis magnitude and 100ps pre-emphasis duration.

Figure 5.19: Pre-emphasis magnitude sweeping on 3.2Gbps and PRBS-31 signal.

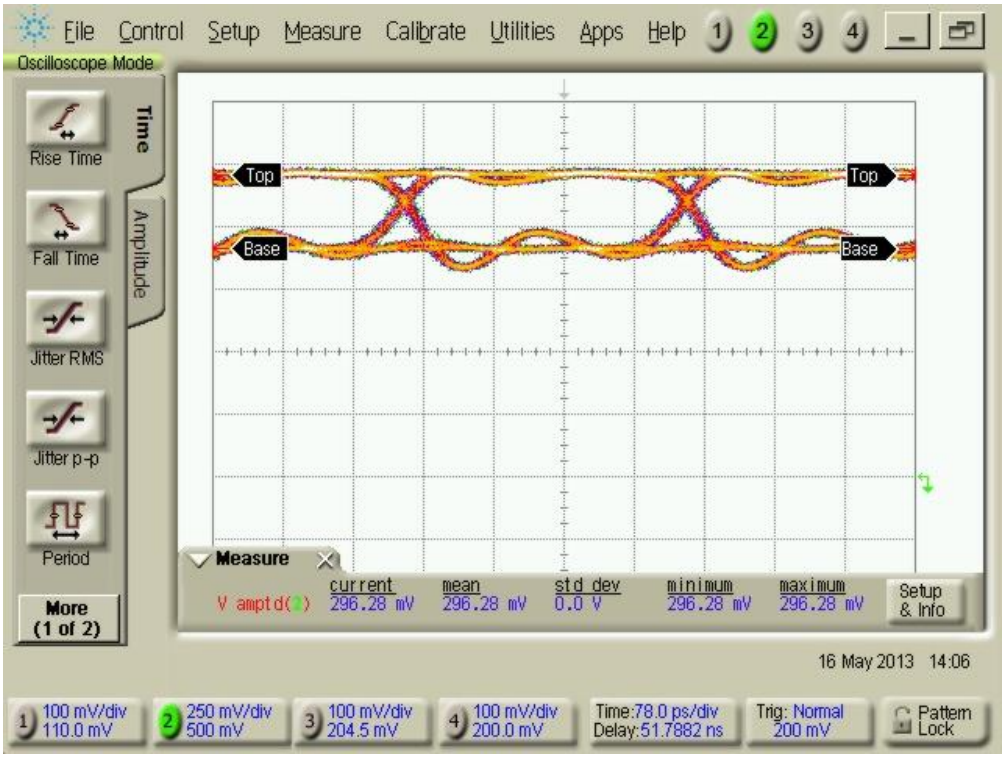
In some lossy environment like long transmission line on PCB board, the emphasis level will help reduce the data distortion for high-speed transmission. The measurement in this section have proved that the pre-emphasis flexibility that the PE board can offer to overcome the difficulties of transmit Gbps signal (especially >3.2Gbps) on PCB board.

5.4.1.2 Voltage swing control

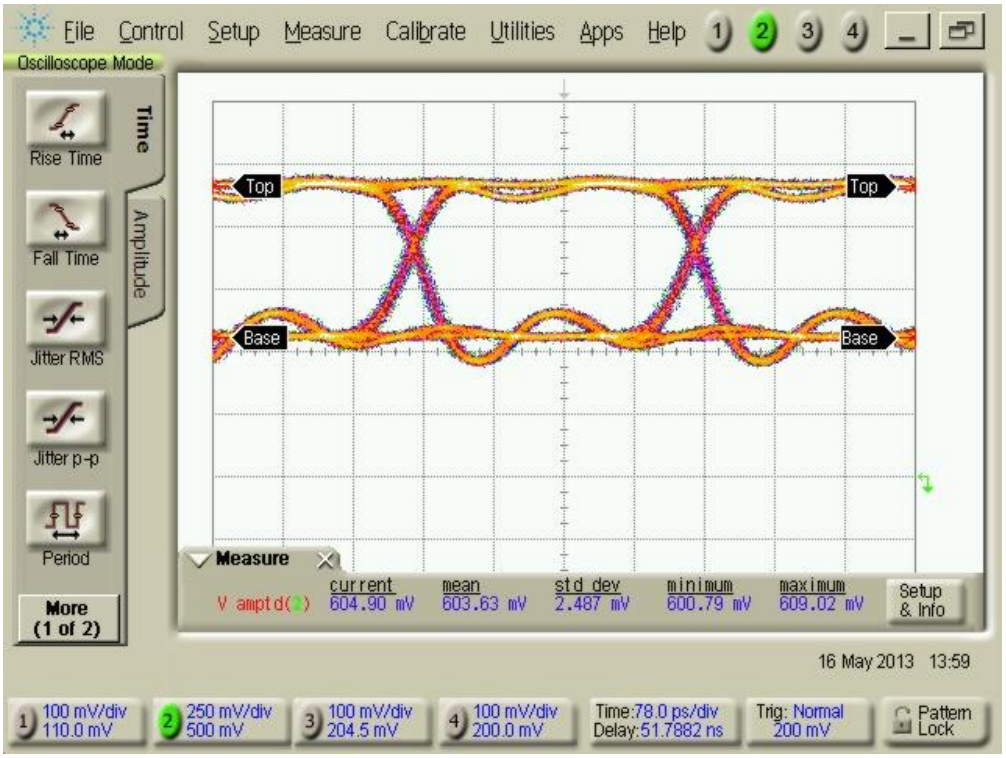
In this measurement, we fixed the pre-emphasis magnitude level at 10% and duration at 100ps (the optimal setting we have figured out at 3.2Gbps in previous test) and changed the single-ended voltage swing from 100mV to 900mV, which are shown in Figure 5.20. Here the DC offset was set to 1.25V (LVDS common voltage). Time base of the scope was fixed to 78ps/div (quarter of 312.5ps), and the voltage base was fixed to 250mV/div. The ground level was set to the second grid line from bottom. In the figures we can see the signal remains a clear eye even at small amplitude, which indicates the TX side of PE board can be applied to test receiver sensitivity of existing I/O standards.



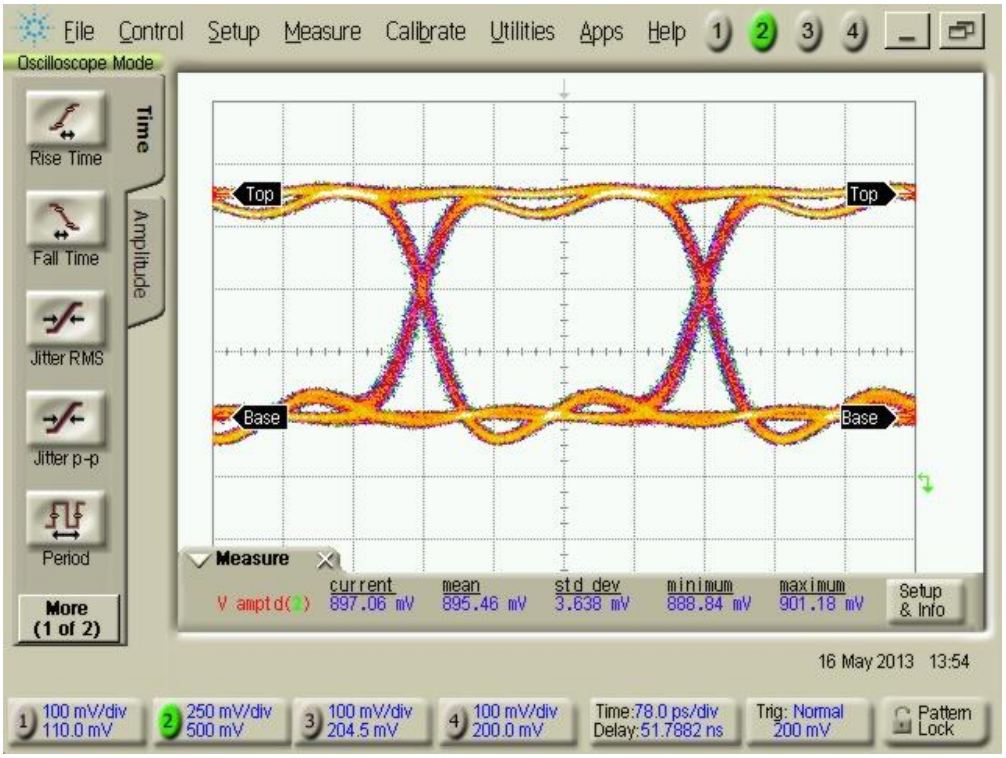
(a) 100mV peak to peak swing.



(b) 300mV peak to peak swing.



(c) 600mV peak to peak swing.

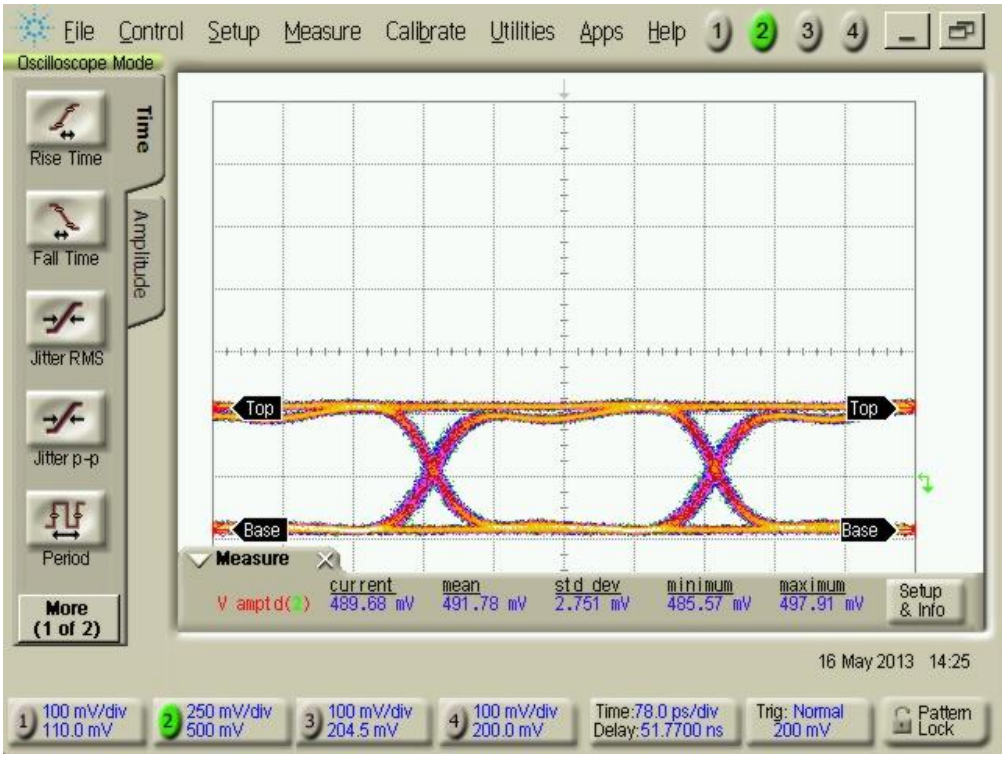


(d) 900mV peak to peak swing.

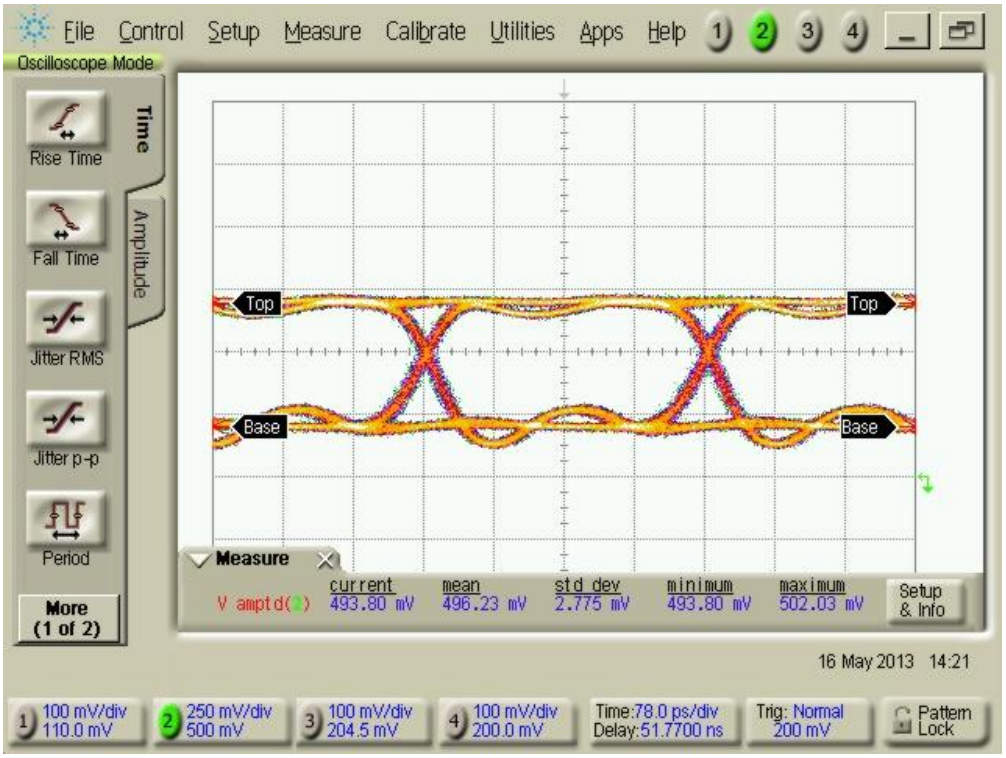
Figure 5.20: Single-ended amplitude sweeping on 3.2Gbp and PRBS-31 signal.

5.4.1.3 DC offset adjustment

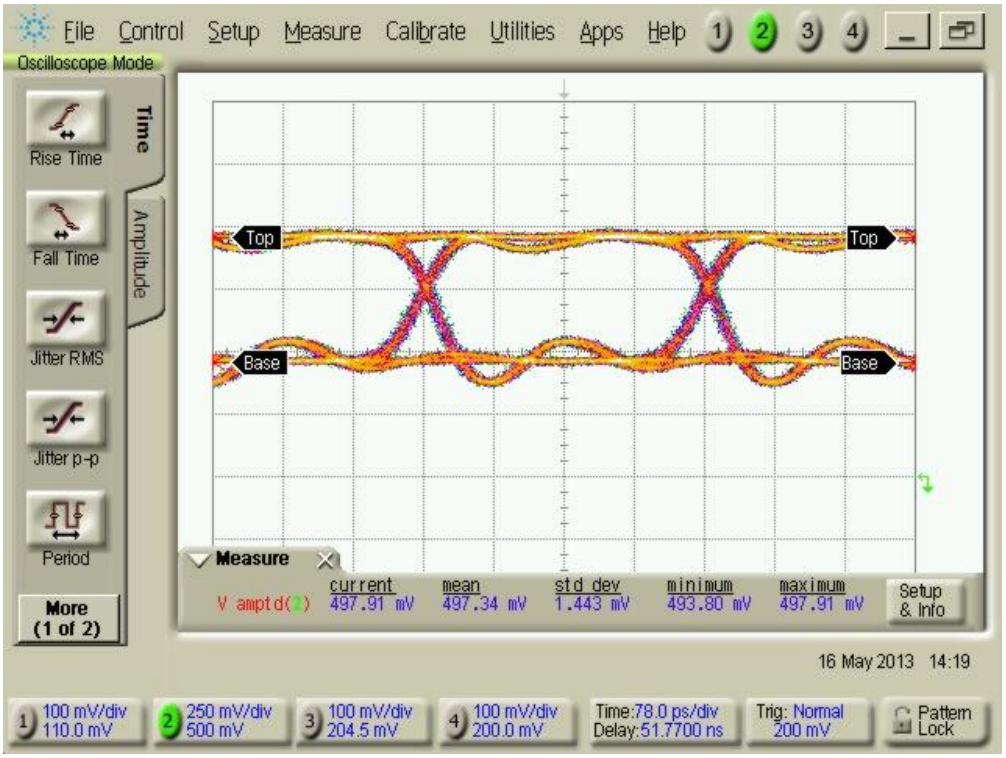
In this section we will show the flexibility of different DC offset settings that the PE card can provide. We fixed the single-ended voltage swing to 500mV (medium swing) and the pre-emphasis setting remained the same at 10% magnitude and 100ps duration as before. The FPGA was still programmed to 3.2Gbps data rate with PRBS-31 pattern. Time base of the scope was fixed to 78ps/div (quarter of 312.5ps), and the voltage base was fixed to 250mV/div. The ground level was set to the second grid line from bottom. Figure 5.21 shows the example of DC offset at 0V, 500mV, 750mV and 1000mV. The ability of producing different DC level in these figures, combine with the amplitude adjustment in previous section, it is persuasive that the PE board output is very flexible and can be used to test different I/O standards (with different DC level) in the market.



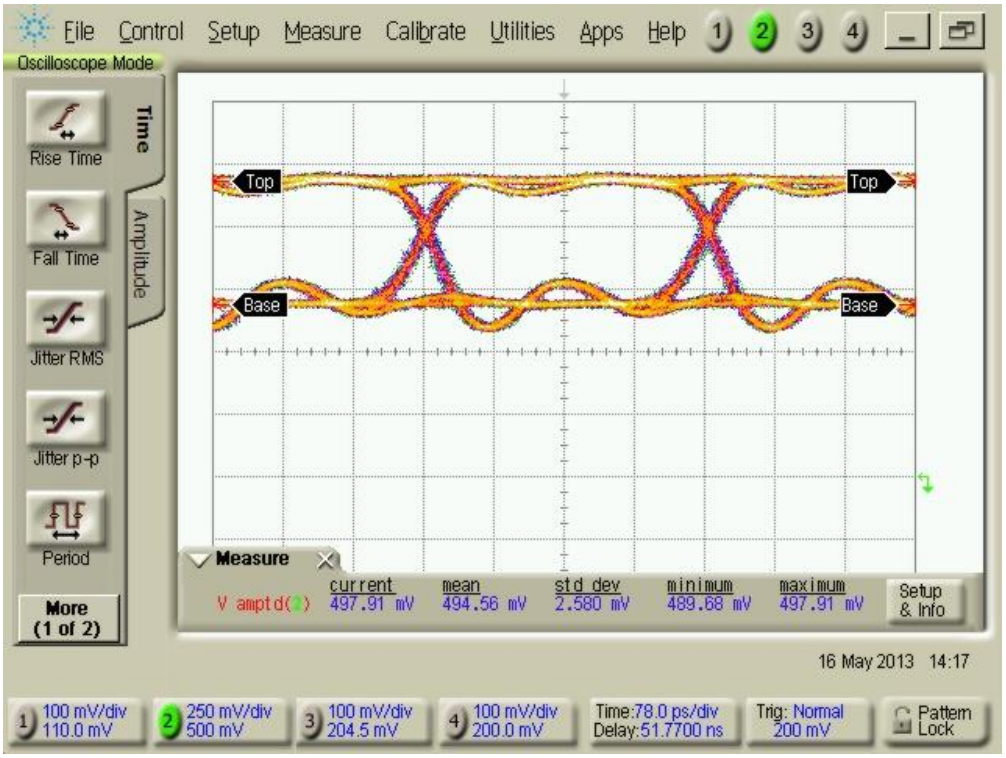
(a) 0V DC offset.



(b) 500mV DC offset.



(c) 750mV DC offset.



(d) 1000mV DC offset.

Figure 5.21: DC offset sweeping on 3.2Gbp and PRBS-31 signal.

5.4.1.4 Crosstalk-Adjacent channel coupling

In Figure 5.22, we show the measurement of adjacent signal coupling ("crosstalk"). Crosstalk is most likely to happen between adjacent signals on the 4-channel PE card because of the close proximity of the transmission lines that carry these high frequency signals. Furthermore, in the PE4 design channels 1 and 2, (as well as 3 and 4), are routed through a common relay. Therefore crosstalk between channels 1 and 2 (or 3 and 4) should exhibit the worst-case crosstalk. To measure this effect, we setup an experiment to drive Channel 1 with a full-amplitude (1000mV) signal (the "aggressor") while monitoring Channel 2 that serves as a "quiet" line. The two channels are connected through coaxial cables to separate channels of the 86100D scope. The active (aggressor) signal is shown in the top waveform, using a 500mV/div scale. The measured crosstalk signal from Channel 2 is shown on the bottom waveform using a 5mV/div scale. This was measured to be only about 16mV peak to peak, which is about 1.5% of the aggressor signal swing.

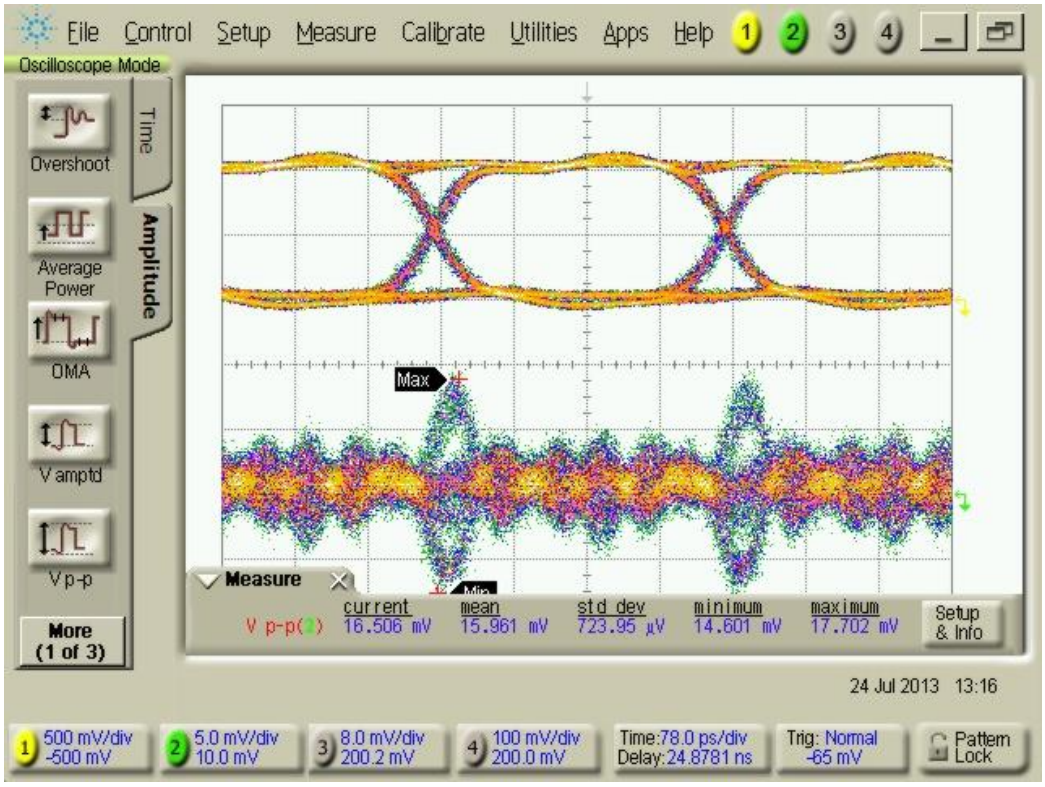


Figure 5.22: Crosstalk measurement on Channel 1 (top, aggressor) and 2 (bottom, quite) at 3.2 Gbps from PE board. Channel 2 only with 8mV peak to peak crosstalk.

5.4.1.5 Receiver testing- Basic Function Verification

In this section, we use TX signal and loopback to RX side to test the receivers on the PE board. The TX side was program to a typical setting with 3.2Gbps, PRBS-31 data 100mV single-ended voltage swing and optimal emphasis (10%, 200ps). The other end of receiver input was 50ohm terminated to ground. The primary receiver used a reference voltage (from LT2656 with a programmed voltage) to recover the loopback signal. In Figure 5.23, channel one (yellow) is the input data of the primary receiver and channel two (green) is the output from the primary receiver. Note that this primary receiver was not well-decoupled, therefore the output signal was relatively noisy at that time. However, the output of well-decouple version of the receiver is shown in Figure 5.24, which we get a very-clean recovered data eye as the input signal does. The result is convincing the receiver works well at this data rate.

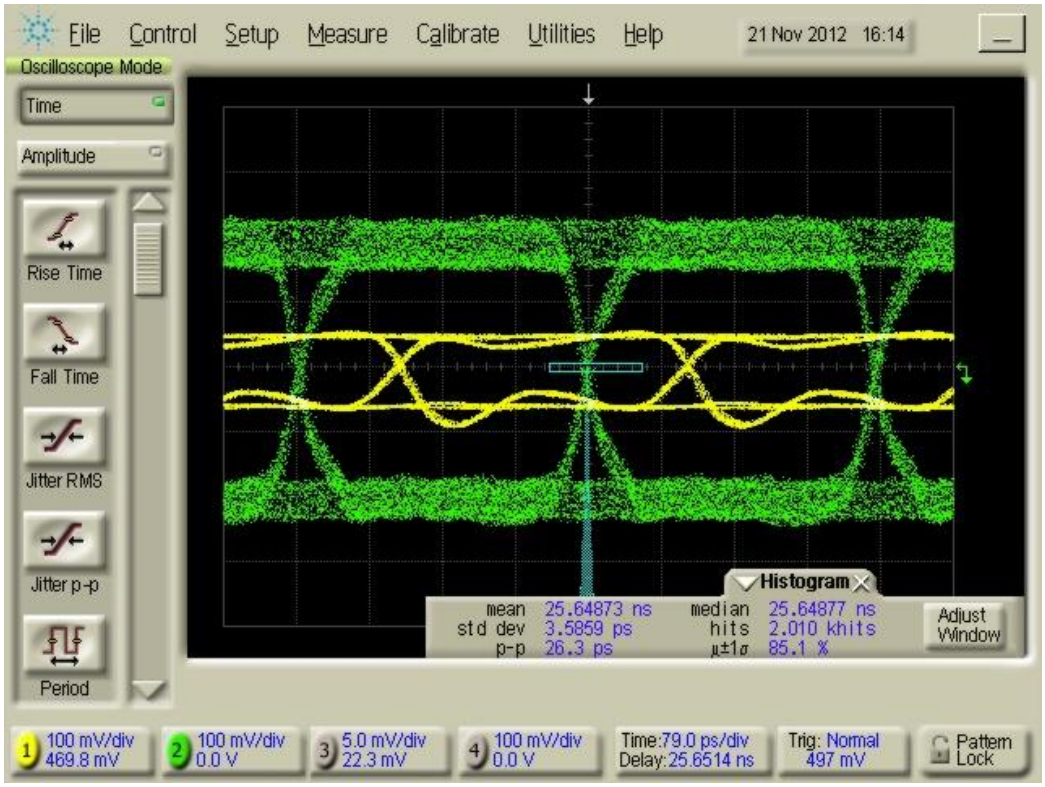


Figure 5.23: The recovered signal (green) vs. the loopback input signal (yellow). The input signal is 3.2Gbps with PRBS pattern and 10%, 100ps emphasis. The recovered signal is performing the same data as the input. Time base = 78ps/div.

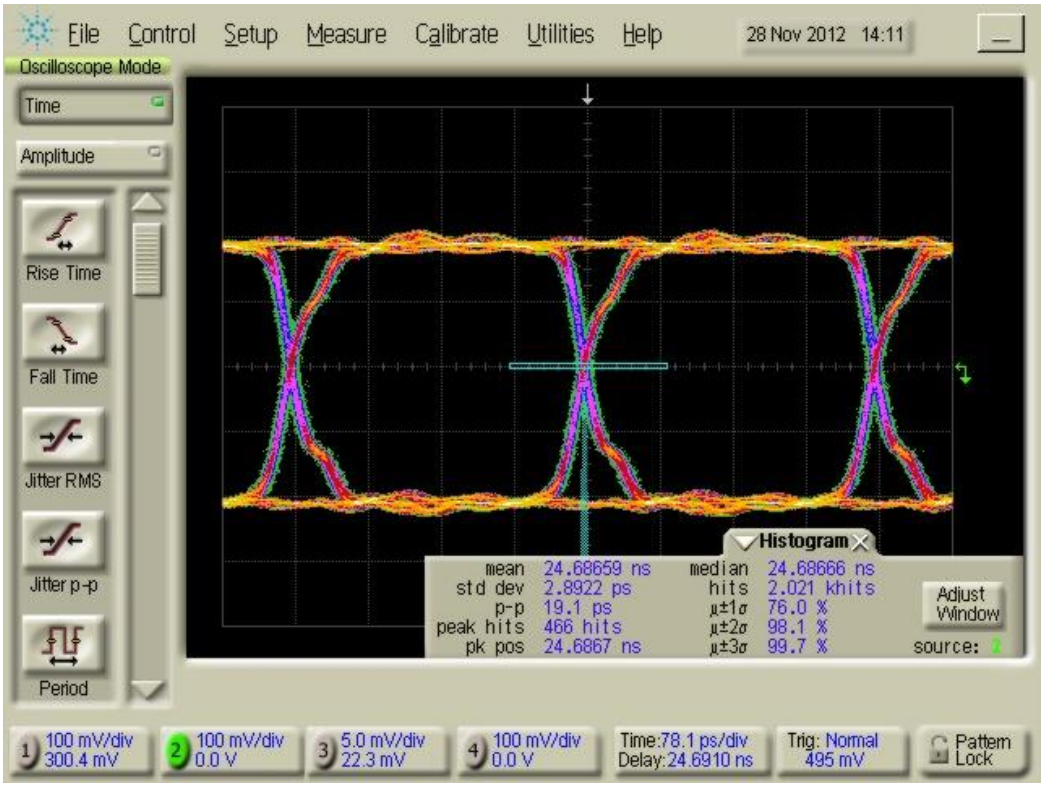


Figure 5.24: The recovered signal with well-decoupled receiver. The input signal is remaining 3.2Gbps with PRBS pattern and 10%, 100ps pre-emphasis. The recovered signal is performing the same data as the input. Time base = 78ps/div.

5.4.1.6 Receiver testing- Data recovery and Eye-monitoring

The primary receiver (HMC674) works with the shadow sampler (HMC874) is able to perform a full data reconstruction in FPGA. The primary receiver basically collect the digital information of the signal, which is the data pattern ("0" or "1"), and the shadow sampler use the shadow clock to sample and collect the analog information of the signal (i.e. rising/falling edges or glitch). This process is to fully reconstruct the waveform and is called eye-monitoring. In Figure 5.25 we demonstrate use of the shadow sampler to capture the data eye (as generated by the PE Driver) at 3.2Gbps. For this measurement we used a 500mV amplitude with 10% magnitude/100ps duration pre-emphasis. The time resolution was 8.5ps and the voltage resolution was 10mV per pixel. In Figure 5.26 we again use the Shadow sampler to measure the Driver eye. However, in this case we increased the Driver

pre-emphasis to 25% amplitude and 200ps duration in order to improve the eye height. Compare these two eyes, we can observe that the eye with 10% pre-emphasis has more blue area on at top-left corner. Also according the to the driver test, we have the knowledge that less pre-emphasis magnitude will occur the rising edge not achieve Vmax immediately. The left side of each figure was capturing the signal with zero delay and the right one was capturing the signal with mandatary 156.25ps (half-UI) delay. These reconstructed eye-diagrams shows the high sensitivity of eye-monitoring which is able to detect the even a small difference on the waveforms.

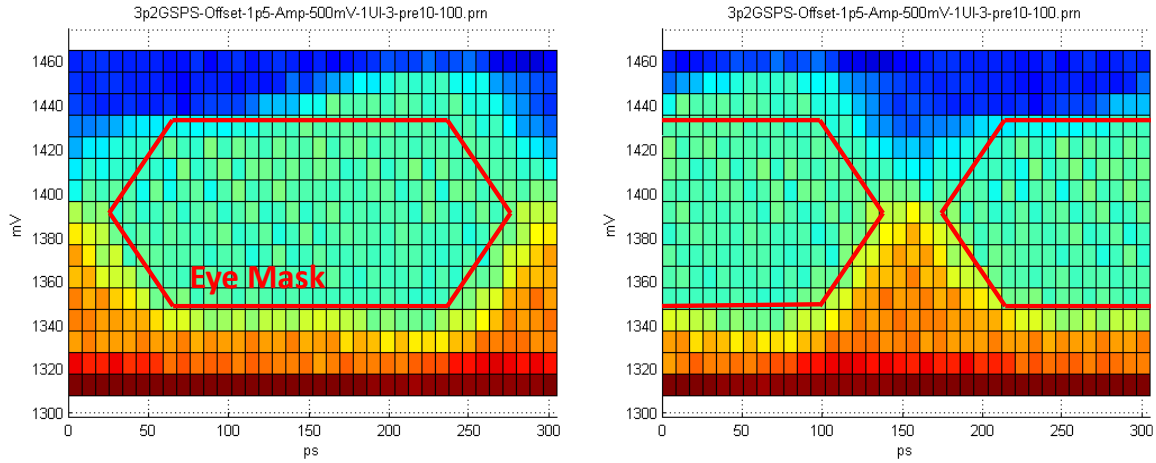


Figure 5.25: Eye-monitoring of 3.2Gbps loopback signal (10% pre-emphasis).

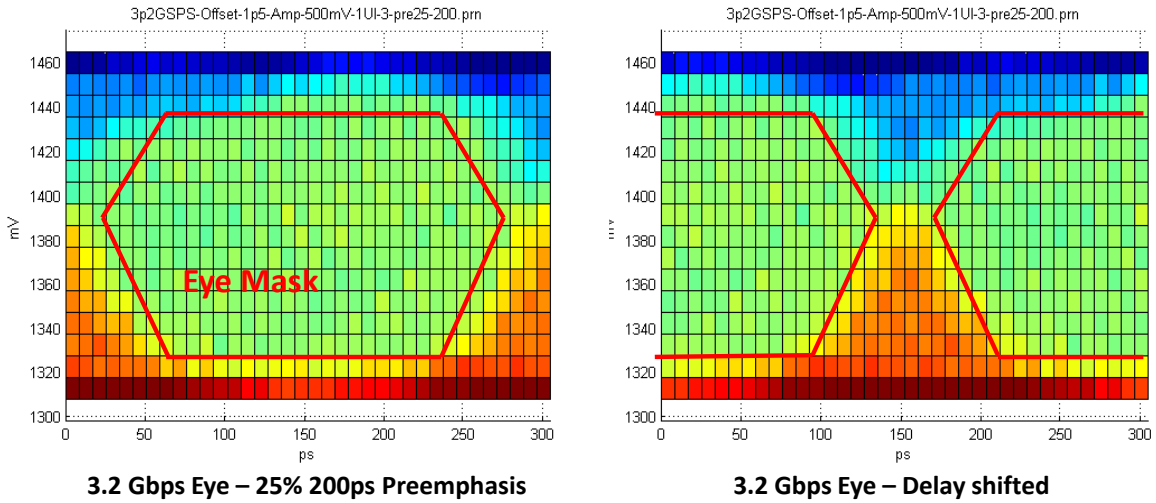


Figure 5.26: Eye-monitoring of 3.2Gbps loopback signal (25% pre-emphasis).

5.4.1.7 The limitation of Driver performance

While the original PE board requirements were to develop the capability to test up to 3.2Gbps, the system has potential to support even faster rates. Figure 5.27 shows the PE Driver output at 5.0Gbps, which is nearly at its maximum rate (with clear eye and reasonable jitter performance). All core components (including Driver, Comparators, Relay, Connectors, Kintex-7) support data rates above 6.4Gbps, the bottleneck of this PE board is actually limited by the programmable delay IC (spec to work at 3.2Gbps). In this demonstration, we actually overclocked the Delay IC to perform 5Gbps signal. While we can see observable jitter increases at the edge of data transition, it is obvious that we don't have too much margin of pushing the PE board to higher speed. Therefore, if a higher data rate pin electronic measurement is required in the future, we need to replace the delay IC.

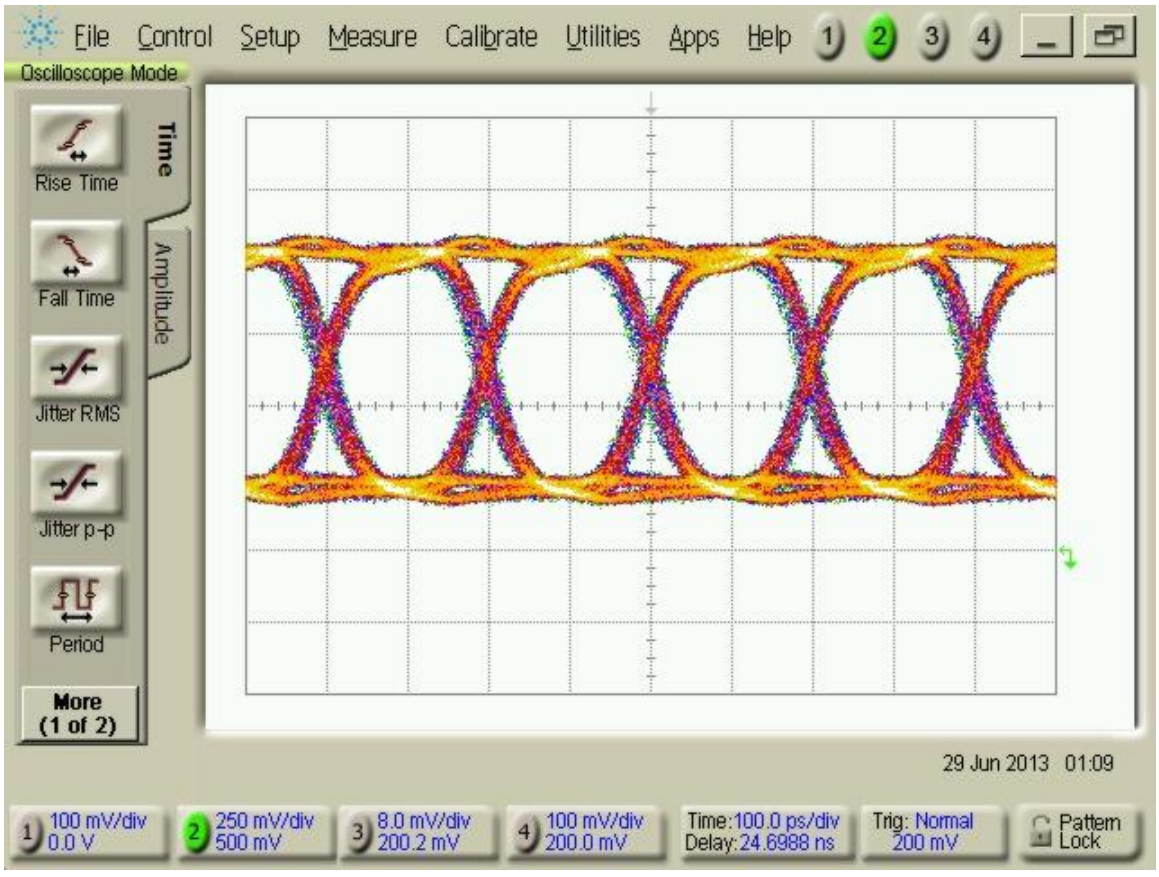


Figure 5.27: Data eye diagram at 5.0Gbps from PE board (10% 100ps pre-emphasis) with 750mV swing and 1.2V DC offset. Time base is 100ps/div.

5.4.2 Ultra-High-Speed Testing Board

In this section, the UHS board takes four channels of GTX signal and serialize to higher data rate. In the other word, this board add another serializing stage to four 10Gbps serial bit streams. The FPGA is programmed to output four-channel and half-clock signals to be synthesized with high-speed serial clock (>10GHz) from Keysight E8257D 40GHz signal generator to generate up to 40Gbps signal. The reference clock of the FPGA is from Keysight 81133A with 1ps RMS jitter. All the waveforms are measured Keysight 81600D scope with 54752A 26/50GHz module (the 50GHz mode is selected to achieve the best result) in this section. The scope is triggered by the divide-by-2 output clock from the MUX chip. The 26.5GHz-bandwidth SMA connectors/cables are also use for the connection of reference clock and triggering clock, the better 40GHz-bandwidth 3.5mm connectors and cables are used at final output to be connected with the scope.

5.4.2.1 Performance

First of all, we will present the basic serializing performance of this UHS board. To confirm the functionality, a slower setup was preferred in the very beginning. The FPGA was initially programmed to output 4-channel 5Gbps PRBS-31 data, and with the 10GHz sampling clock from 40GHz signal generator, resulting a 20Gbps PRBS-31 differential signal is generated and shown in Figure 5.28. The rise/fall time was around 15ps (80%-20%), the RMS jitter was 1.3ps and the peak-to-peak jitter was about 10ps. Consider to the equipment limitation including the jitter from 81133A (~1ps) and the jitter added by FPGA GTX and the MUX chip, this result is very brilliant. With the similar approach, we can present 36Gbps (the spec of the MUX) signal with four channels of 9Gbps signal and 18GHz sampling clock in Figure 5.29. A 40Gbps signal generated from four channels of 10Gbps signal and 20GHz sampling clock is shown in Figure 5.30. The RMS and peak-to-peak jitter at these two data rates increased slightly to 1.5ps and 11.5ps respectively, and the rise/fall time remained ~13ps which was very close to the specification.

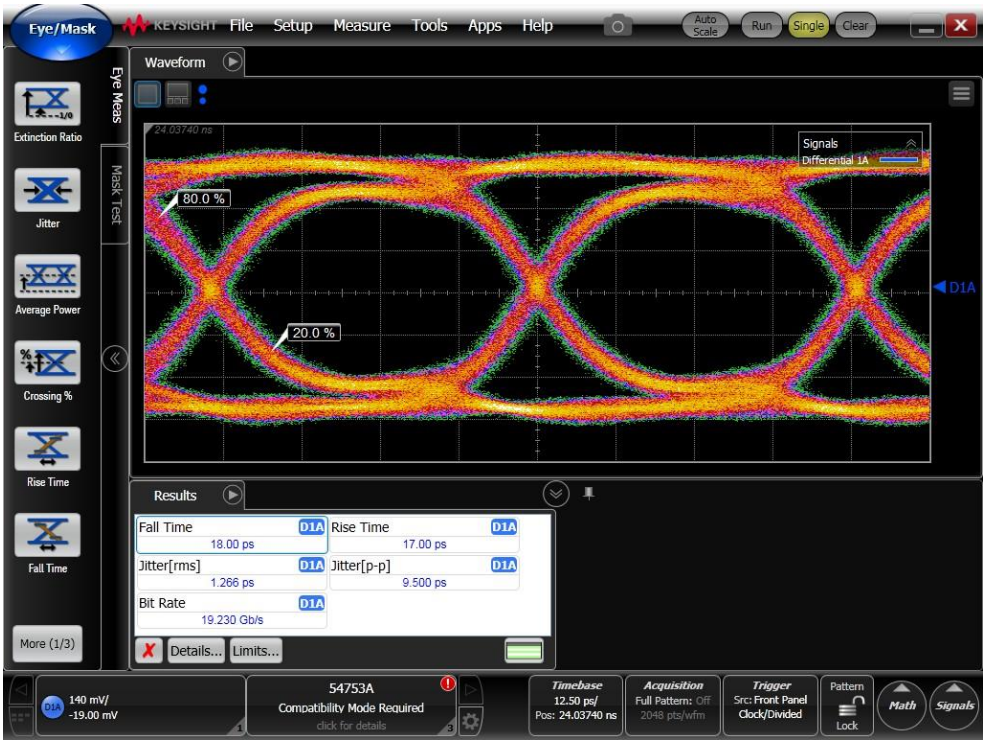


Figure 5.28: 20Gbps differential signal with PRBS-31pattern. Time base is 12.5ps and voltage swing is ~950mV.

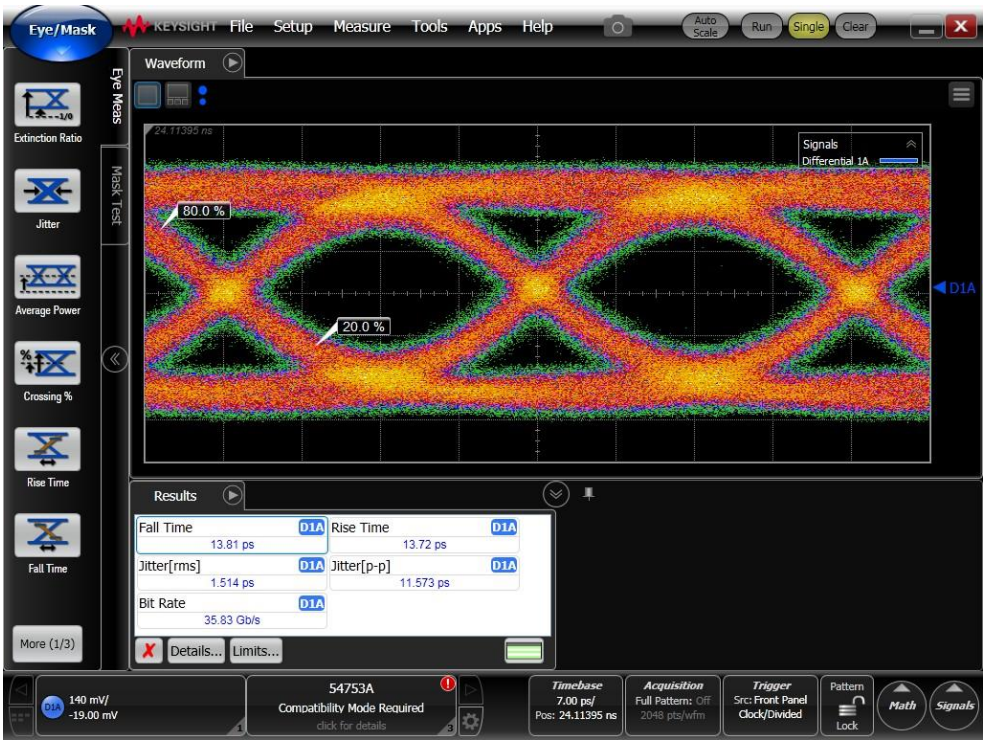


Figure 5.29: 36Gbps differential signal with PRBS-31pattern. Time base is 7ps and voltage swing is ~950mV.

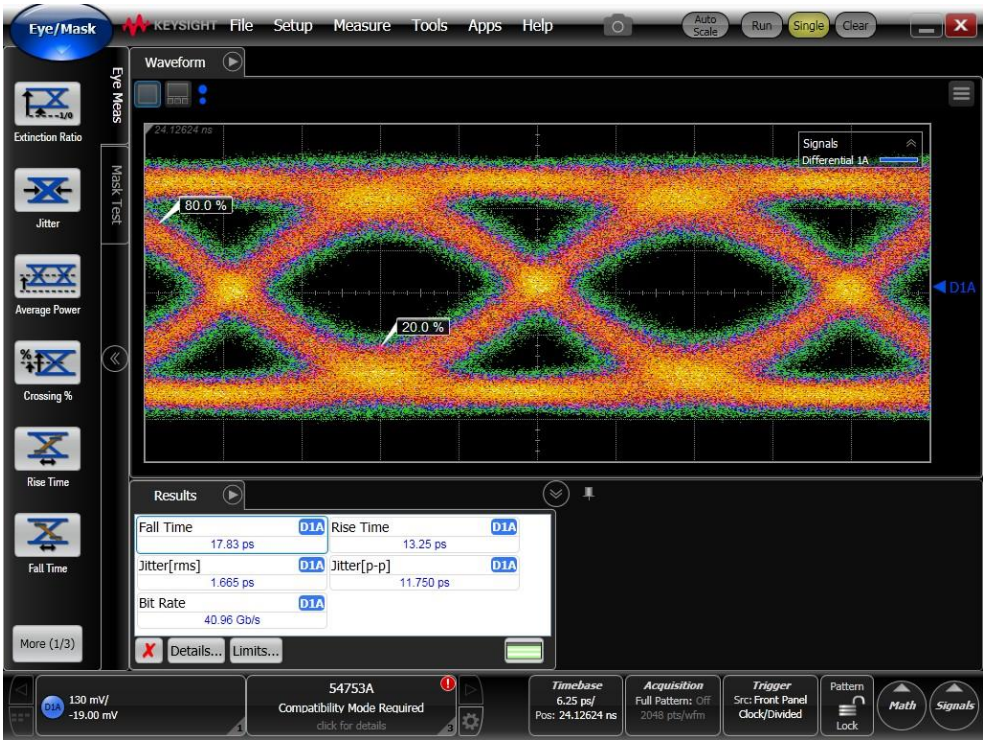
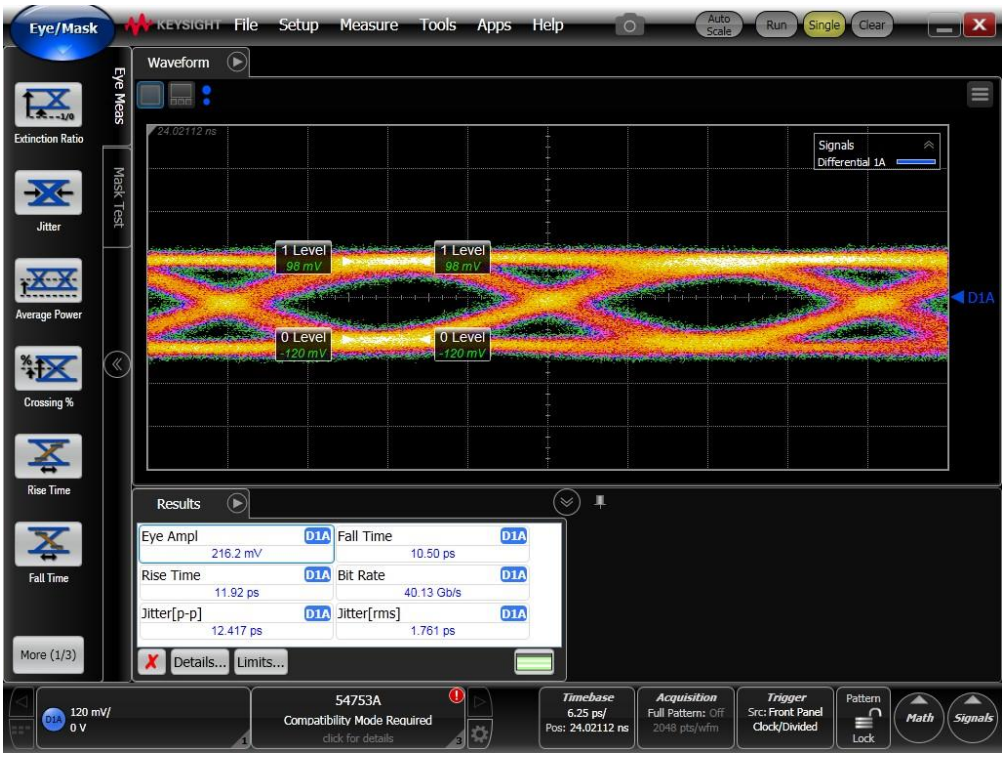


Figure 5.30: 40Gbps differential signal with PRBS-31pattern. Time base is 6.25ps and voltage swing is ~950mV.

The reason why the jitter increases at high data rate is mostly due to the bandwidth of triggering input of the 86100D scope. The specification shows that the bandwidth of the scope is 13GHz with 1.2ps jitter injection. The divided-by-2 clock coming out of MUX chip is very close to the bandwidth limit. Also the bandwidth of measuring plugin module (50GHz) and the SMA/SMP/3.5mm (26.5GHz~40GHz) cables are the other potential limitations for the measurement. Therefore it is believable that these are the best results under the current setup, a better result might rely on higher-resolution equipment such as the precision time-base triggering module and 70GHz bandwidth sampling module.

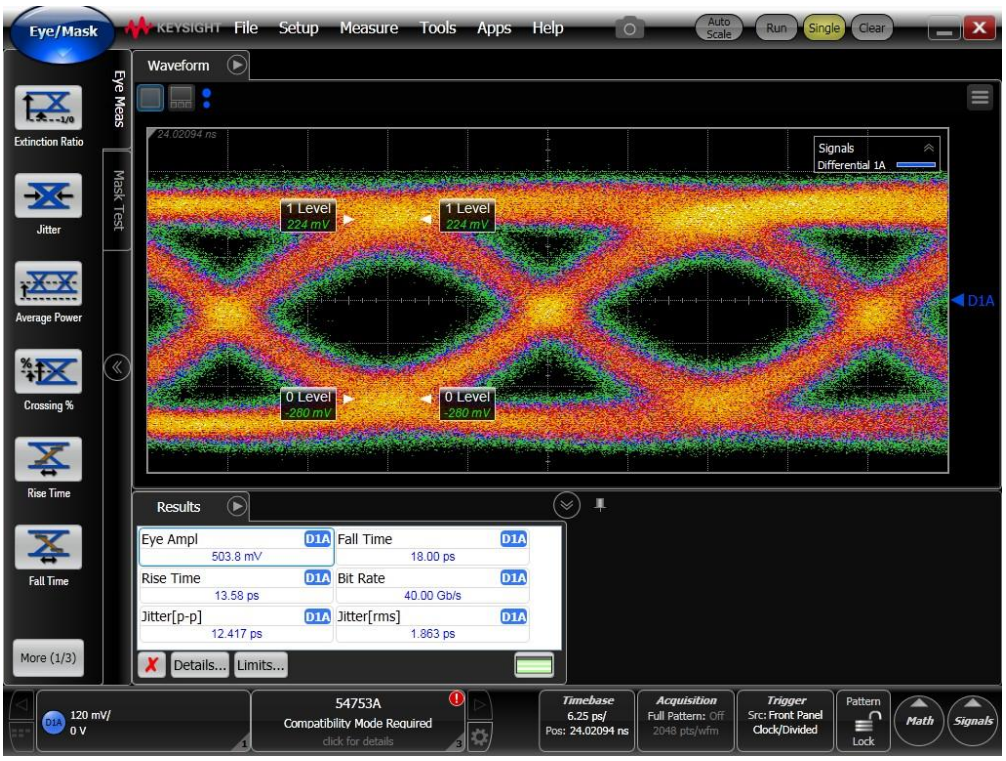
5.4.2.2 Output amplitude adjustment

The MUX chip offers amplitude adjustment by the voltage control pin. In Figure 5.31, we programed the FPGA and the extension board to output 40Gbps signal and swept the differential amplitude to 250mV (a), 400mV (b) and 650mV (c) respectively.



(a) 250mV (6.25ps/div, 120mV/div)

(b) 400mV (6.25ps/div, 120mV/div)



(C) 650mV (6.25ps/div, 120mV/div)

Figure 5.31: Extension board amplitude adjustment at 40Gbps

5.4.2.3 Output duty-cycle adjustment

In Figure 5.32, a duty cycle adjustment is performed. Again we kept the data rate to 40Gbps and the pattern remained to PRBS-31, the differential amplitude was programed to 600mV (peak to peak). By programming the voltage control bit on the driver we are able to get a non-50% duty cycle signal, this also indicates that the driver is able to run at an even higher data rate. A clear-open eye with about 20ps width is shown on the right side, which means there is a potential for this part to run at 50Gbps. To form a 50Gbps signal indicates four 12.5Gbps signals and 25GHz sampling clock are needed. However, the multi-channel GTX is not able to run at this data rate and the maximum frequency of the delay chip is limited at 24GHz, plus the MUX is only spec to run at 36Gbps, which means all the devices on UHS board and FPGA-based Test Platform are all running out of specification. Also the limitations from equipment we have mentioned will become more

significant at higher data rate. Therefore we don't have the ability of present the 50Gbps signal now and the related work is reserved to be realized in the future.

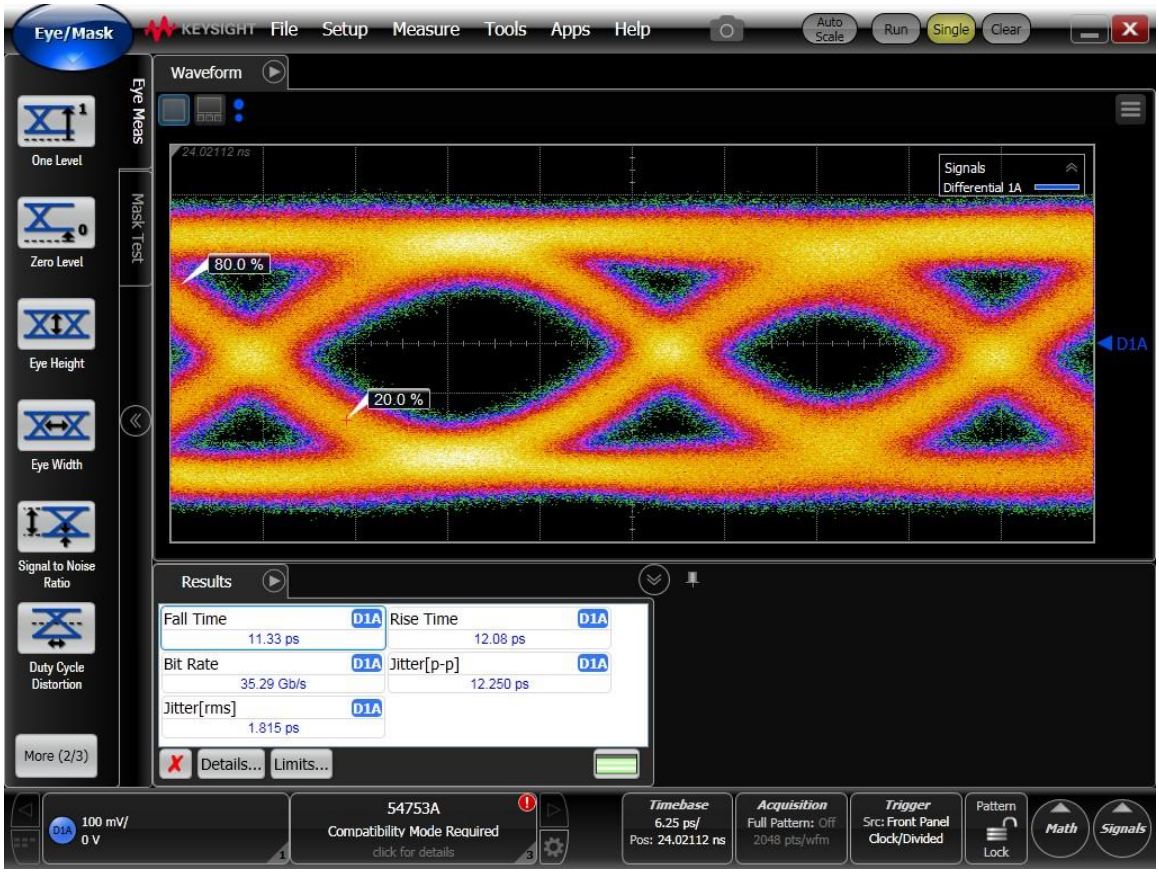


Figure 5.32: Duty-cycle adjustment at 40Gbps

5.5 Summary

In this chapter, we demonstrate a pin electronic testing board and an ultra-high-speed testing board on the FPGA-based Testing Platform. The TX side of the PE board offers the flexibility of sweeping pre-emphasis, voltage swing and DC level which is not fully provided by the FPGA main board. The RX side provides the minimum of 10mV data detection and an accurate data-reconstruction ability which is not widely seen in other testing equipment. The UHS testing board extends the product life of FPGA-based Testing Platform by increasing the speed of output signal in FPGA from 10Gbps to 40Gbps. These two boards both show the most significant advantage of the FPGA-based Test platform has over ATE systems, which is the low module-updating cost. By building the extension modules with low cost and updated FPGA firmware, the function and the performance of FPGA-based Testing Platform is enhanced without replacing the core hardware (which is considered to be relative expensive on this testing platform). In short, the realization of the specific-purpose extension modules makes this testing platform a higher chance to replace the ATE systems and change the world of digital testing.

CHAPTER 6

CONCLUSIONS

6.1 Summary

The object of this research is to develop a low-cost, adaptable testing platform for multi-GHz digital applications to replace the conventional ATE systems by using FPGA approach. In Chapter 1 we have illustrated the limitations of high-speed digital testing, and the motivation of why we want to do this research. Since most advanced ATEs are very expensive, this sophisticated equipment is not always available for testing commercial digital devices which are cost-sensitive. Combining with the drawbacks of high power consumption, and very-limit and costly updating modules, the ATEs become the obstacle for testing the most cutting edge design. Therefore we need to design a more economical testing platform that can easily adapt to new testing requirements without incurring excessive development or huge capital costs. In Chapter 2 we first reviewed the development of modern ATEs, and also BIST method that is frequently used in digital testing. Few critical limitations were also found in these two testing methods. In the last part of Chapter 2, the FPGAs with high flexibility and economical cost have the potential to be developed into an ideal testing system as most designers desire for years.

In Chapter 3 we have introduced the ATG board to extend the ATE performance into GHz range, also the ATG board provides an excellent function of letting user to define the data pattern and each individual edge to perform full ability of “timing-on-the-fly” which is not seen in the market. The 45nm Spartan-6 FPGA in the ATG works as an extended digital controller and still requires the host of the ATE system. The well-developed ATG algorithm is able to generate the control signals for the high-speed components to synthesize the arbitrary waveform. The algorithm offers a very flexible method to define a continuous, no dead-zone and on-the-fly timing waveform. The HS

section has provided a precise “8-to-1” serializing to form a 3.2Gbps signal from eight 400MHz phase-adjust reference clocks and eight 400Mbps control signals. This FPGA-based ATG module has not only shifted the performance of ATE testing method to a higher level but also proved the high potential of using FPGA in digital testing. However, this ATG cannot operate individually as an independent testing platform and needs to rely the host of ATEs. Therefore, these existing issues make the ATG module still hard to fully replace the role of ATE systems in semiconductor industry.

To solve this issue we have implemented a FPGA-based Testing Platform in Chapter 4 and also developed the auxiliary extension board of this platform in Chapter 5. In Chapter 4, a 28nm Kintex-7 FPGA was used to realize the core of this testing platform. The LUTs inside FPGA can be implemented to different testing algorithms and the I/Os on the FPGA package offer DUT testing ability. The 400 normal I/Os can be used for moderate-speed test (<1Gbps) and the low-speed control ports and communication interfaces of the FPGA and auxiliary chips implemented on the platform. The 16 high-speed GTX channels provide up to 10Gbps TX/RX differential signals which offer this platform the most advanced digital testing ability in this industry. Despite of the powerful FPGA core, four high-speed and multi-pins slots allows new extension boards to be plug-in on the main board to enhance the function of the existing platform. The two specific extension boards were designed and presented in Chapter 5. The PE board provides typical 3.2Gbps data transmission and receiving. The TX side provides pin-electronic testing features such as flexible voltage swing, pre-emphasis and DC level adjustment for DUT. The RX side is able to perform as low as 10mV data detection with the full capability of waveform reconstruction. The UHS testing board extends the FPGA GTX signals from current main-stream standards to possible future applications. The 40Gbps TX/RX bi-directional data transition ensures this platform to catch the pace with the development of semiconductor route. The FPGA-based Test Platform and extension modules finally become a competitive candidate to replace those ATE systems

The above discussion gives us a brief review of the works that has been done in this research. In the rest of this chapter, we will first focus on the improvement of the current designs and the more related works can be done in the future. In the last section the major contributions will be illustrated to conclude this research.

6.2 Future works

6.2.1 Multi-GHz Arbitrary Timing Generator

Due to the development of semiconductor, the performance of this ATG might become very limited in the future. Therefore we need to figure out a solution to extend the product life of current ATG. In the ATG experiment section, we used either burst or MUX mode to achieve higher data rate (>3.2Gbps). However, as we mentioned before, for these two methods we need to limit the flexibility and the most powerful capability of ATG, which is to generate continuous, no dead zone and timing-on-the-fly signal. In order to provide complete capability of ATG to higher speed testing, few critical bottlenecks with possible solution are illustrated in this section.

6.2.1.1 Improvement for Max Data Rate

The first bottleneck of ATG is the algorithm requires heavy-loading calculations inside FPGA. Xilinx Spartan 6 is an economical 45nm device which can operate the ATG algorithm at 200MHz/400Mbps and update the fine delay control signals every 2.5ns. Although this algorithm has been deeply pipelined and optimized to get better performance, this frequency is still almost the limit for Spartan-6 to run such complicated calculations. The post-route simulation from Xilinx ISE software reported the speed limitation of running the algorithm is about 235MHz/470Mbps. Therefore if we want to implement a higher performance ATG based on the same HS logic architecture, the FPGA need to

output higher than 400Mbps and each edge-generator/FF pair requires to run higher than 400MHz. Running faster than 400MHz is not a big issue for high-bandwidth devices (>1GHz) in HS section. However, >400Mbps output data rate means the relative low-speed FPGA needs to refresh the delay values and control signals less than 2.5ns, this is very challenge for the Spartan 6 FPGA. To eliminate this issue, using the next-generation FPGA device (28nm or better) to replace current Spartan-6 FPGA is the easiest approach to achieve higher throughputs of generating those delay values.

Despite of increasing FPGA output data rate, the other possible solution is increasing the bit-width for serializing. Now the algorithm generates 8 parallel timing sets for serializing in HS section. If we can expand the algorithm to generate 16 sets and also increase the number of hardware edge-generator/FF pairs simultaneously, we will be able to double data rate (from 3.2Gbps to 6.4Gbps) with full ATG function. However, the possible issue of this approach is to increase another stage of XOR-serializing, which might add more jitter to the output signal. To solve this unavoidable issue, less-jittery components (i.e. the XOR device which was used for MUX mode) might be required to improve the jitter performance of whole system. The other possible issue is the expanded algorithm might run out of logic resource inside FPGA and I/O pins on the package, therefore a FPGA with larger logic capacity and package is needed. This method requires both hardware replacement and algorithm update, but also has higher potential to push the performance of current architecture to higher level.

6.2.1.2 Timing resolution, jitter and rise/fall Time

In the current ATG we chose to use 10ps timing resolution because it was readily available in the programmable delay chip, and also because it represents about 3% of the target bit period (at 3.2Gbps). However, even finer resolution is desired, especially if this method is extended to 6.4Gbps or faster. With this in-mind, we included provisions in the prototype design to add a 5ps control bit to each of the delay generators.

Jitter is the primary concern since it accounts for most of the residual timing error that exists following calibration. In the prototype ATG, the total jitter is made up of random jitter (RJ) and deterministic jitter (DJ). The random component of jitter is dominated by the RJ of the master reference clock, which we obtained from the host ATE. This accounts for about 4ps RMS jitter. Each of the ECL delay chips adds about 1ps to RJ. Deterministic jitter is a cumulative value that comes from data-dependencies of the SiGe flip-flop and XOR gates in the high-speed section, each contributing 2-4ps to DJ. Since the hardware architecture itself doesn't have too much room to improve the RJ, a lower-jitter reference clock source could be used in the future to reduce RJ.

The other important issue is the rise/fall time of the serializing XOR array. The rise/fall time of the XOR we use is specified at 20ps and the operating data rate is 13Gbps, it looks like there is still a lot bandwidth for current ATG. However, if we want to increase the data rate (especially >10Gbps) in the future, the current rise/fall time performance starts to get limited. In the burst mode experiment of Chapter 3, when the ATG was programmed to produce 70ps pulse, the relative long rise/fall time (~20ps) of the XOR makes the output signal cannot have full transition (to reach Vmax/Vmin). Therefore, a better rise/fall time device such as HMC744 (~10ps rise/fall time) is needed if we want to apply up to 10Gbps data rate in the future.

6.2.1.3 Applications of ATG algorithm

The ATG algorithm was originally developed as the plug-in module on ATE systems. However, it can be easily applied to other digital testing platform. FPGA-based low cost and high-performance testing platform designed in Chapter 4 is an ideal approach replace traditional expensive ATE. This new designed testing equipment reserves multiple extension slots for future plug-in module. Therefore the interface of this ATG board can be modified and plug-in this platform and become its extension module. Furthermore, this algorithm is realized by Verilog hardware description language, which makes it compatible

to other FPGA or ASIC devices. As we know FPGA is a very flexible components and can be re-configured to different applications. With moderate modifications on the algorithm, it is able to fit on various FPGA components and perform similar work. Therefore another potential application is to program the algorithm to the existing FPGA-based Testing Platform and build HS section as extension module. With this approach, the update only requires expending the current code and renewing the HS module instead of replacing the whole hardware.

6.2.2 FPGA-based Testing Platform and extension boards

The FPGA-Based Testing Platform and the extension boards we have developed have offered a whole new low-cost, wide-band and high-performance platform for digital test. As we mentioned before, the platform has a great potential to be well-developed. Here are the possible future works on this platform.

6.2.2.1 The performance of FPGA transceivers

The core technology of this platform is FPGA, while we were using 28nm Xilinx Kintex-7 family on the current design, the GTX performance is limited to ~12Gbps. Today the more advance 20nm FPGAs (Xilinx UltraSCALE series) [80] start to appear in the market and the production of 16nm FPGA (Xilinx UltraSCALE+ series) [81] is on schedule. With the new semiconductor process, the high-performance transceiver can go up to 32Gbps. The higher data rate means the support of future high-speed standards. The huge upgrading is not only about the performance improvement but also the IP updates for those transceivers. The new IP includes a lot new features like more efficient LUTs, phase control at TX output end, wider serialization mode (8-byte) for higher data rate, new encoding/decoding support (120B/130B), improved PRBS generator and checker, more steps of pre/post-emphasis configuration, and the higher flexibility of configuring PLL (fractional mode support) [82] [83] [84]. These new features have enhanced the ability of

FPGA and allow the users to design and define new methodology for more testing scenarios.

6.2.2.2 The bandwidth of main connectors

The other possible issue of limiting the performance of the system is the connector bandwidth of four extension slots. The four extension slots bring the high-speed signals from FPGA to the extension board. If the FPGA is upgraded to a newer version, the 10GHz-bandwidth Samtec connector is becoming the bottleneck of the platform. For the experiment of UHS extension board in Chapter 5, we used four 10Gbps signals to synthesize a 40Gbps signal. However, the speed of these signals passing through the connector are almost on the margin of the bandwidth that the specification claims. This constraint also limited our experiment of pushing the UHS board to output 50Gbps in the experiment section of Chapter 5. Under current architecture, if we want to replace the FPGA core with higher-performance version, a >10GHz bandwidth connector is definitely required. There are few 28Gbp+ connectors are already provided by Samtec [85] with almost the same features compare with current connector (100+ I/O channels, separated power pins), these connectors can be adopted as we implement the next version of FPGA-based Test Platform.

6.2.2.3 Pin electronic testing at 6.4Gbps

Although 3.2Gbps is still the main-stream speed of the most I/O standards, some newer I/O standards with higher data rate is existing or on the way to the market. Therefore the PE board might not be applicable for the future testing. According to the PE board architecture, all the component (driver, relay and two receivers) on both TX and RX path is able to transmit/capture 6.4Gbps (or even 10Gbps) signal except the delay chip (spec at 3.2Gbps). The delay chip was overclocked to run at 5.0Gbps in our last experiment in Chapter 5, but this speed is about the limit and there is no much margin to be pushed more.

However, we still want to keep the skew management on the PE board, therefore a high performance delay IC is required if we want to realize 6.4Gbps PE testing. Fortunately, we have discovered a high-speed delay chip HMC911 as we developed the UHS board. This delay chip with 24GHz bandwidth should give us enough margin for >6.4Gbps pin electronic test.

6.3 Contributions and Conclusion

The objective of this research was to develop a high-speed and low cost testing platform by using FPGA technology. The experimental results in this thesis have demonstrated a great competition on FPGAs over the advance ATE systems. Therefore these results provide a strong evidence that the approach we have presented is able to replace the ATEs and completely change the method we had used to test the digital devices for the past few decades. Furthermore, this approach has shown the high upgradability and broad applications to future testing scenarios.

The major contributions of this thesis are interpreted as follows: The ATG provides a low-cost solution to extend the traditional ATE to run at multi-Gbps speed. The developed algorithm for ATG allows users to define edge information and data pattern with fine resolution (~10ps) at high data rate (3.2Gbps). This algorithm and combined high-speed logic has realized the capability of unlimited software simulation. This is dreamed by most of digital designers for years and there is still no other testing system in the market can reach the same achievement. The FPGA-based Testing Platform has provided a high-performance and low-cost solution over conventional ATE systems. With the full control from the user end, multi-channel ability and individual power supply, the FPGAs finally can work as the core component instead only an extension module of a testing system. This FPGA-based platform not only supports several hundred moderate-speed channels but also

the fastest I/Os in the industry, which makes it the first time that we can test the high-speed and multi-channel DUT without expensive ATE systems. The power consumption of this platform is very low so that we can get rid of the giant power supplies and liquid-based cooling module we usually suffer from ATE systems. Unlike the fixed ATE systems, the architecture of this platform is very flexible and has a lot potential to be developed for other testing purposes. The example applications such as PE board in this thesis has extended the wider I/O standard supporting range of the main board, and the UHS testing board pushed the platform to support up to tremendous 40Gbps signals. All of above features have helped to solve the dilemma of performance and price we have faced in digital testing and convinced us to deploy this method to more digital applications.

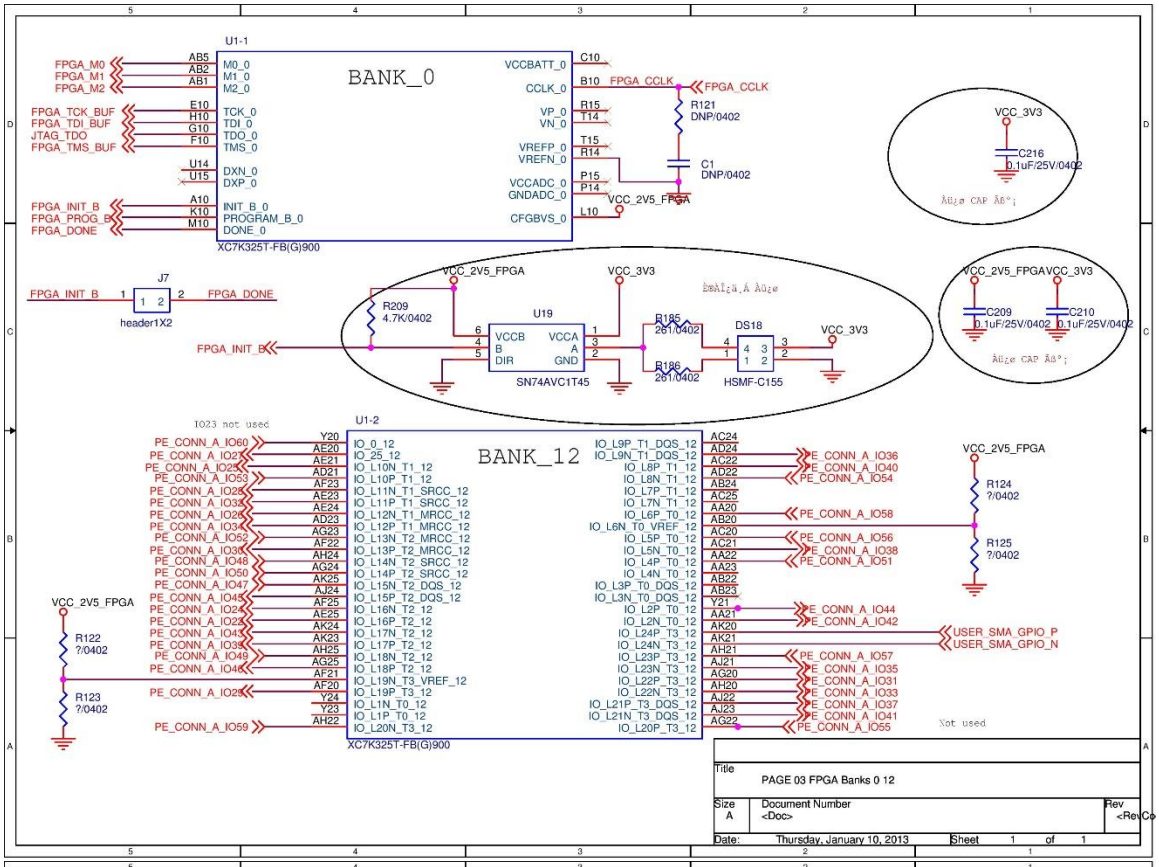
Although the methodologies of this thesis have provided a complete solution to the future digital testing, still there are more to do with this new concept. In the area of digital testing, this approach only shows the initial blueprint and great potential for exploring. The development of semiconductor will not stop in the coming future. In order to completely replace the ATE systems, and also keep the same pace of semiconductor industry, more efforts should be put on this approach and the extended concepts. Therefore the contributions made in this thesis can be maximized and the core concept will be widely adopted by the industry in foreseeable future.

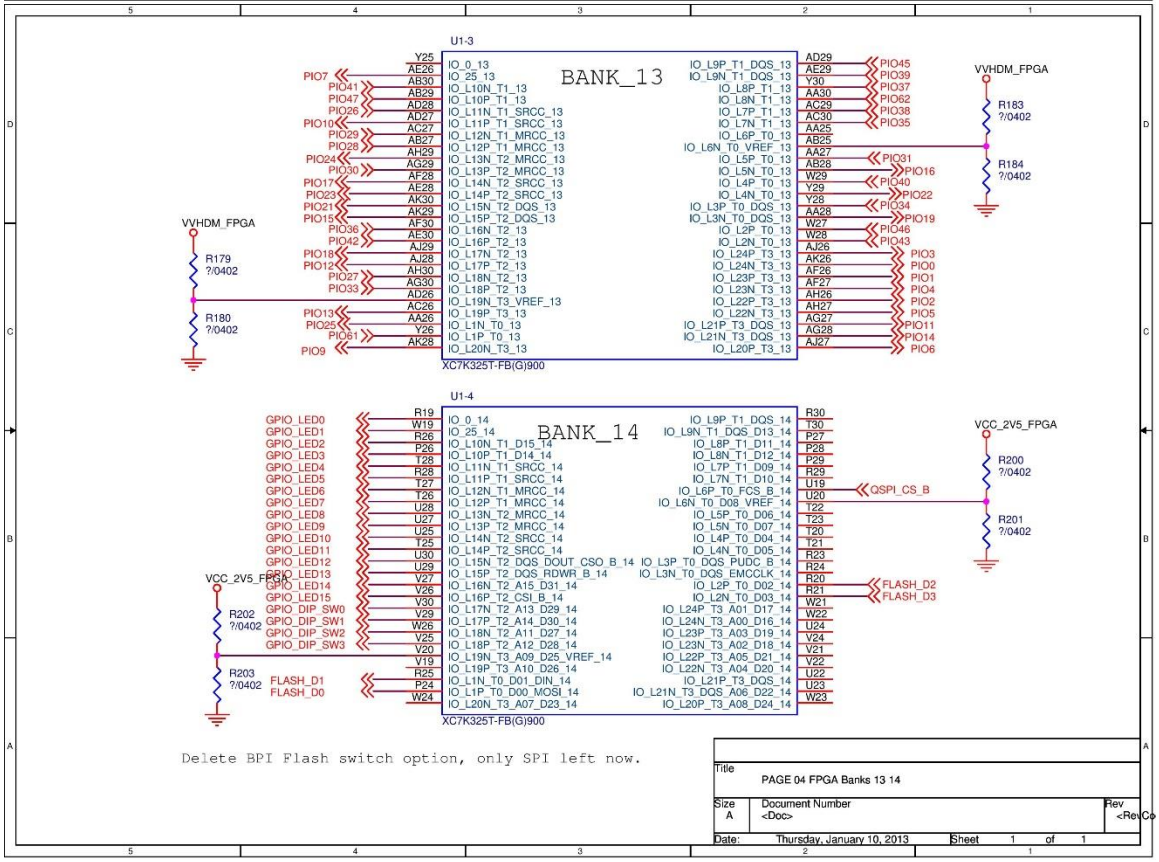
APPENDIX A

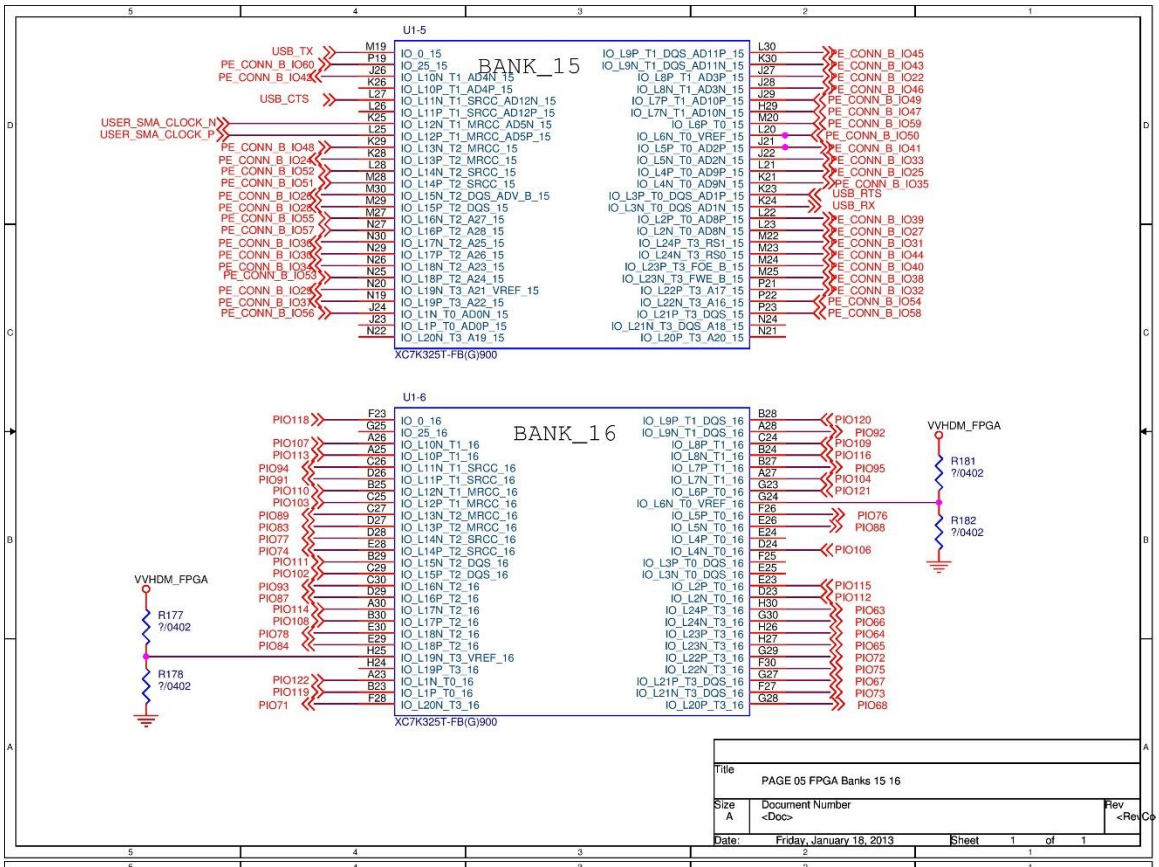
FPGA-BASED TESTING PLATFORM DESIGN

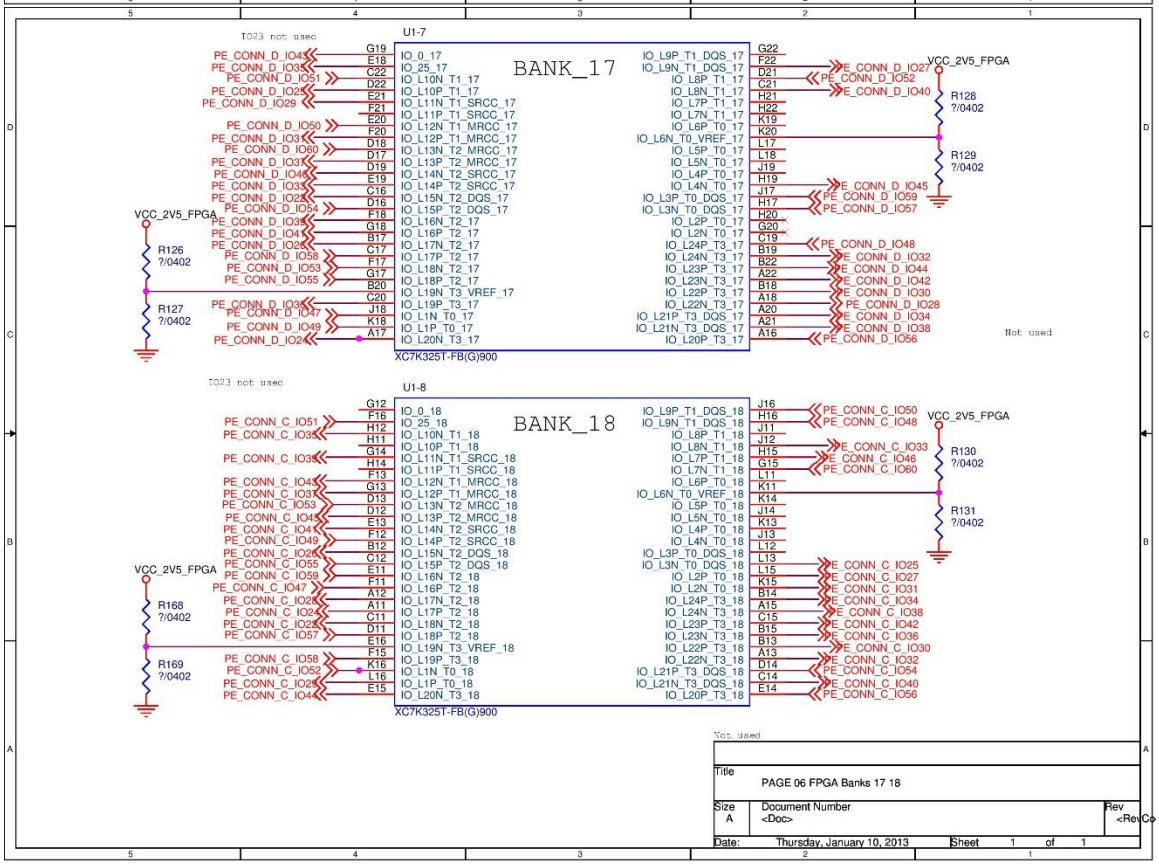
This appendix illustrates schematics design of FPGA-based Testing Platform. The connection of each FPGA bank I/O, power supply connection, control signals, reference clock and high-speed signal connections are well defined in these figures.

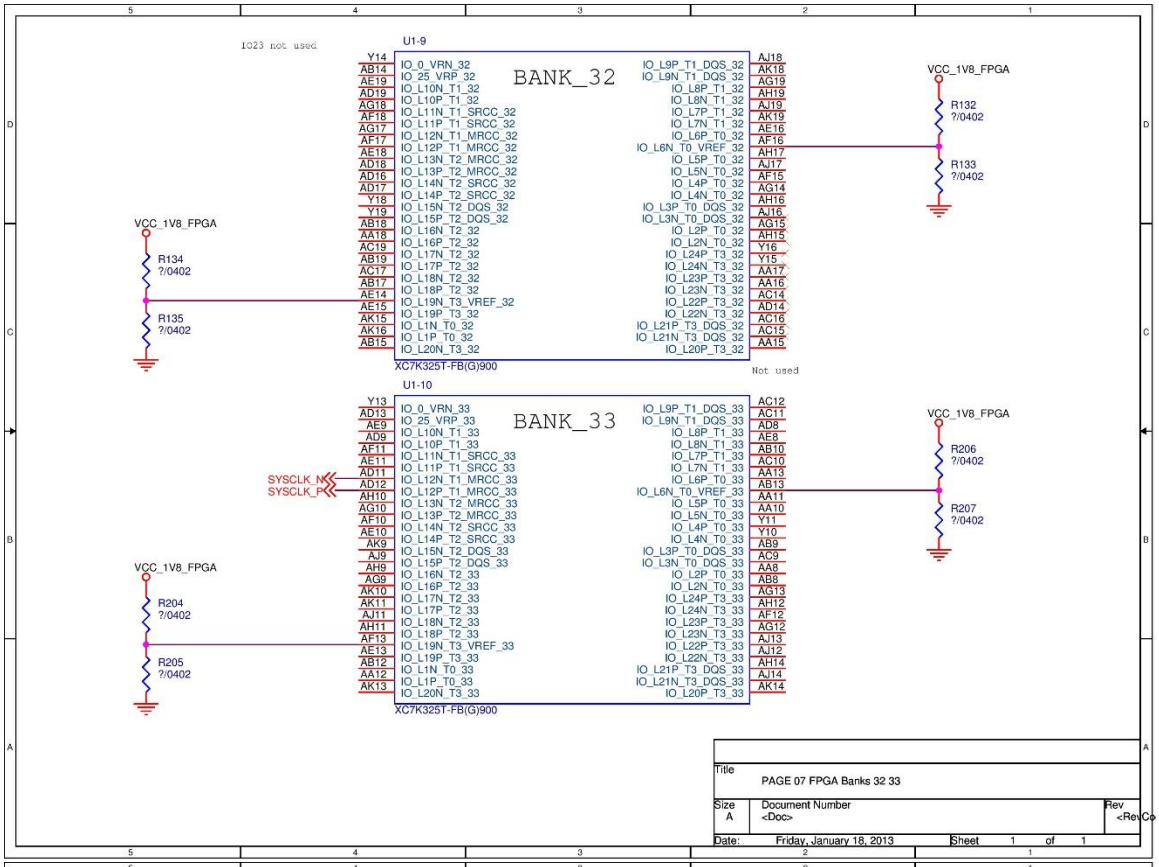
Some critical schematics pages are: The first six pages show the FPGA normal I/O connections, and follows one page of high-speed GTX connections. The power supply and ground connection to FPGA is shown in page 10 and page 11. Those high-speed SMA connectors is shown in page 23 and the other peripheral chips connections are defined in page 26 and page 27. Page 48 to Page51 shows the connection of four Samtec connectors and the rest of schematics page in this appendix are decoupling circuits for power supply, LEDs and dip-switch controls.

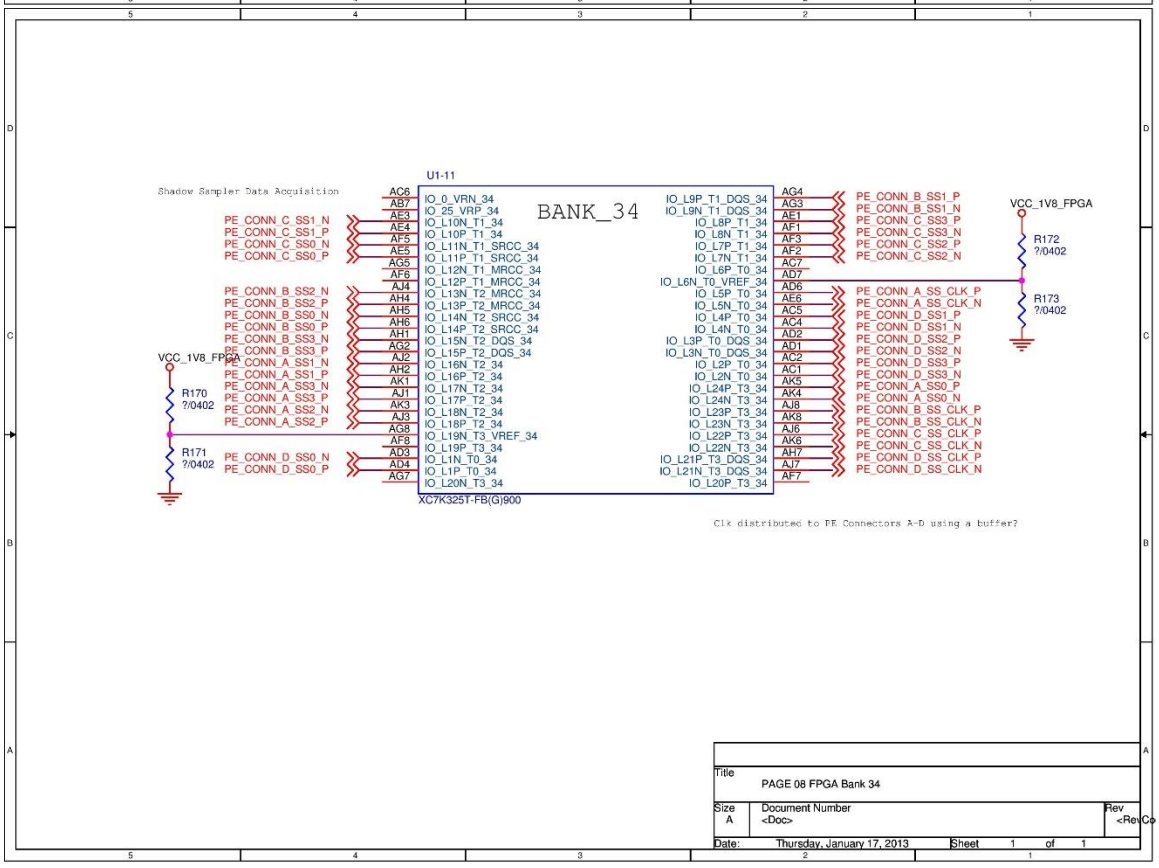

U1-1
BANK_0
XC7K325T-FB(G)900
FPGA_M0
FPGA_M1
FPGA_M2
FPGA_TCK_BUF
FPGA_TDI_BUF
JTAG_TDO
FPGA_TMS_BUF
FPGA_INIT_B
FPGA_PROG_B
FPGA_DONE
VCCBATT_0
CCLK_0
FPGA_CCLK
R121
DNP/0402
C1
DNP/0402
VCC_2V5_FPGA
VCC_3V3
C216
0.1uF/25V/0402
J7
header1X2
R209
4.7K/0402
U19
SN74AVC1T45
R185
261/0402
R186
261/0402
DS18
HSMF-C155
C209
C210
0.1uF/25V/0402
U1-2
BANK_12
XC7K325T-FB(G)900
IO23 not used
R124
?/0402
R125
?/0402
R122
?/0402
R123
?/0402
USER_SMA_GPIO_P
USER_SMA_GPIO_N
Not used
Title
PAGE 03 FPGA Banks 0 12
Size
A
Document Number
<Doc>
Rev
Date:
Thursday, January 10, 2013
Sheet
1
of
1

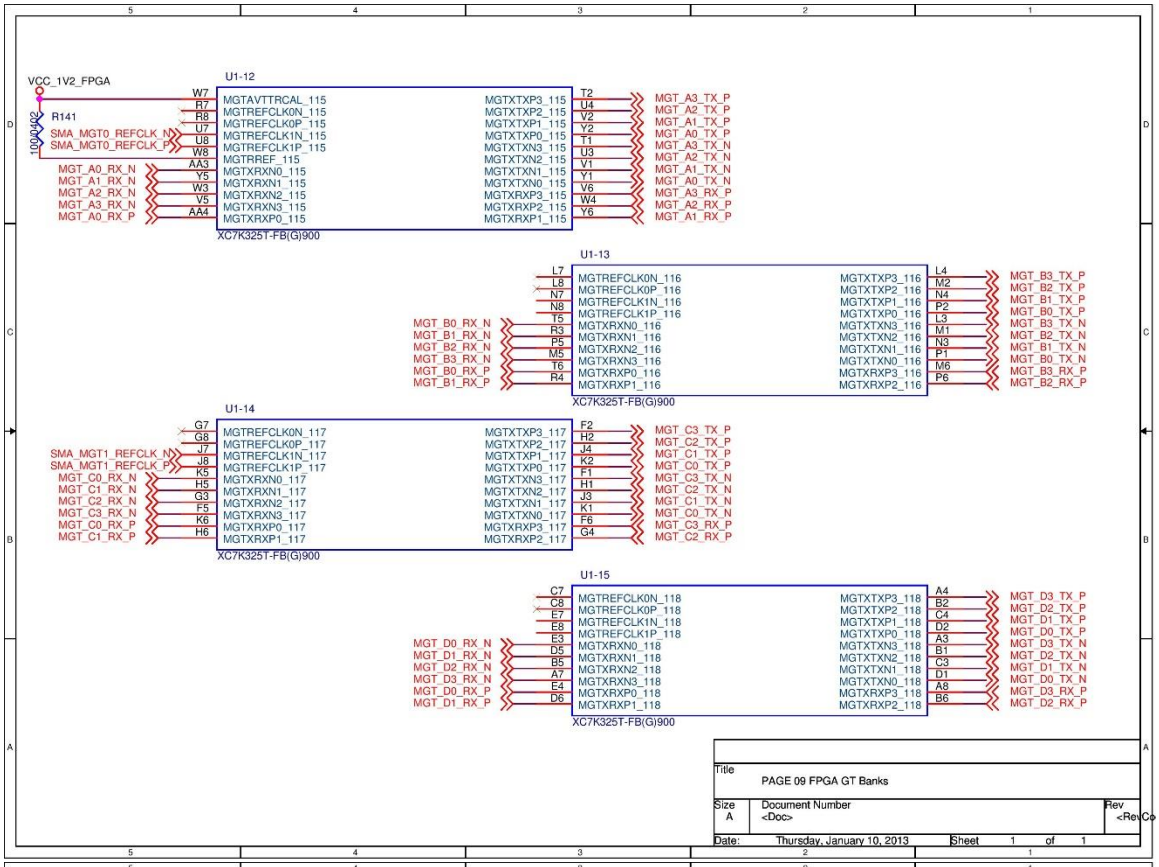

U1-3
BANK_13
XC7K325T-FB(G)900
VVHDM_FPGA
R183
?/0402
R184
?/0402
R179
?/0402
R180
?/0402
U1-4
BANK_14
XC7K325T-FB(G)900
VCC_2V5_FPGA
R200
?/0402
R201
?/0402
R202
?/0402
R203
?/0402
QSPI_CS_B
FLASH_D2
FLASH_D3
FLASH_D1
FLASH_D0
Delete BPI Flash switch option, only SPI left now.
Title
PAGE 04 FPGA Banks 13 14
Size
A
Document Number
<Doc>
Rev
Date:
Thursday, January 10, 2013
Sheet
1
of
1

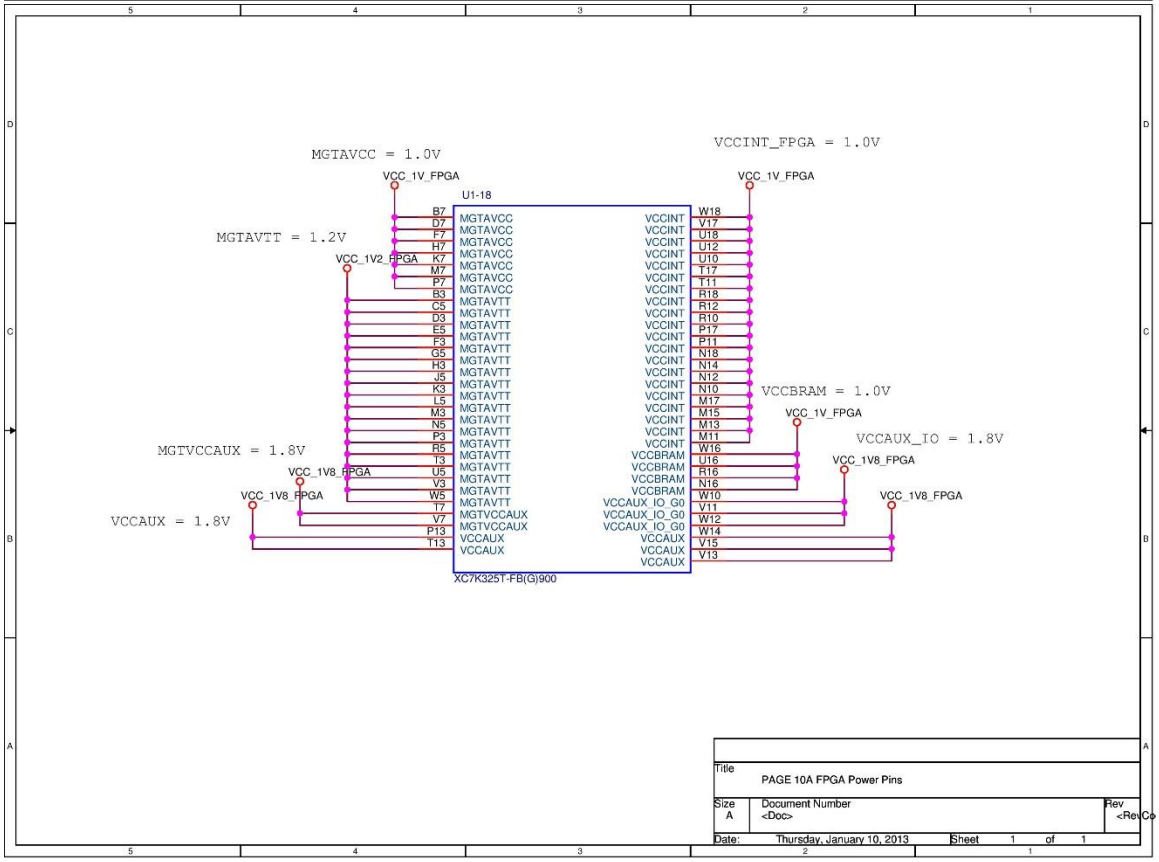

U1-5
BANK_15
XC7K325T-FB(G)900
USB_TX
PE_CONN_B_IO60
PE_CONN_B_IO43
USB_CTS
USER_SMA_CLOCK_N
USER_SMA_CLOCK_P
PE_CONN_B_IO48
PE_CONN_B_IO24
PE_CONN_B_IO52
PE_CONN_B_IO51
PE_CONN_B_IO26
PE_CONN_B_IO28
PE_CONN_B_IO55
PE_CONN_B_IO57
PE_CONN_B_IO36
PE_CONN_B_IO35
PE_CONN_B_IO34
PE_CONN_B_IO53
PE_CONN_B_IO29
PE_CONN_B_IO37
PE_CONN_B_IO54
PE_CONN_B_IO56
PE_CONN_B_IO45
PE_CONN_B_IO43
PE_CONN_B_IO22
PE_CONN_B_IO46
PE_CONN_B_IO49
PE_CONN_B_IO47
PE_CONN_B_IO59
PE_CONN_B_IO50
PE_CONN_B_IO41
PE_CONN_B_IO33
PE_CONN_B_IO25
PE_CONN_B_IO35
USB_RTS
USB_RX
PE_CONN_B_IO39
PE_CONN_B_IO27
PE_CONN_B_IO31
PE_CONN_B_IO44
PE_CONN_B_IO40
PE_CONN_B_IO38
PE_CONN_B_IO32
PE_CONN_B_IO54
PE_CONN_B_IO58
U1-6
BANK_16
XC7K325T-FB(G)900
PIO118
PIO107
PIO113
PIO94
PIO91
PIO110
PIO103
PIO89
PIO83
PIO77
PIO74
PIO111
PIO102
PIO93
PIO87
PIO114
PIO108
PIO78
PIO84
PIO122
PIO119
PIO71
PIO120
PIO92
PIO109
PIO116
PIO95
PIO104
PIO121
PIO76
PIO88
PIO106
PIO115
PIO112
PIO63
PIO66
PIO64
PIO65
PIO72
PIO75
PIO67
PIO73
PIO68
VVHDM_FPGA
R177
?/0402
R178
?/0402
R181
?/0402
R182
?/0402
Title
PAGE 05 FPGA Banks 15 16
Size
A
Document Number
<Doc>
Rev
<Rev.Co
Date:
Friday, January 18, 2013
Sheet
1
of
1

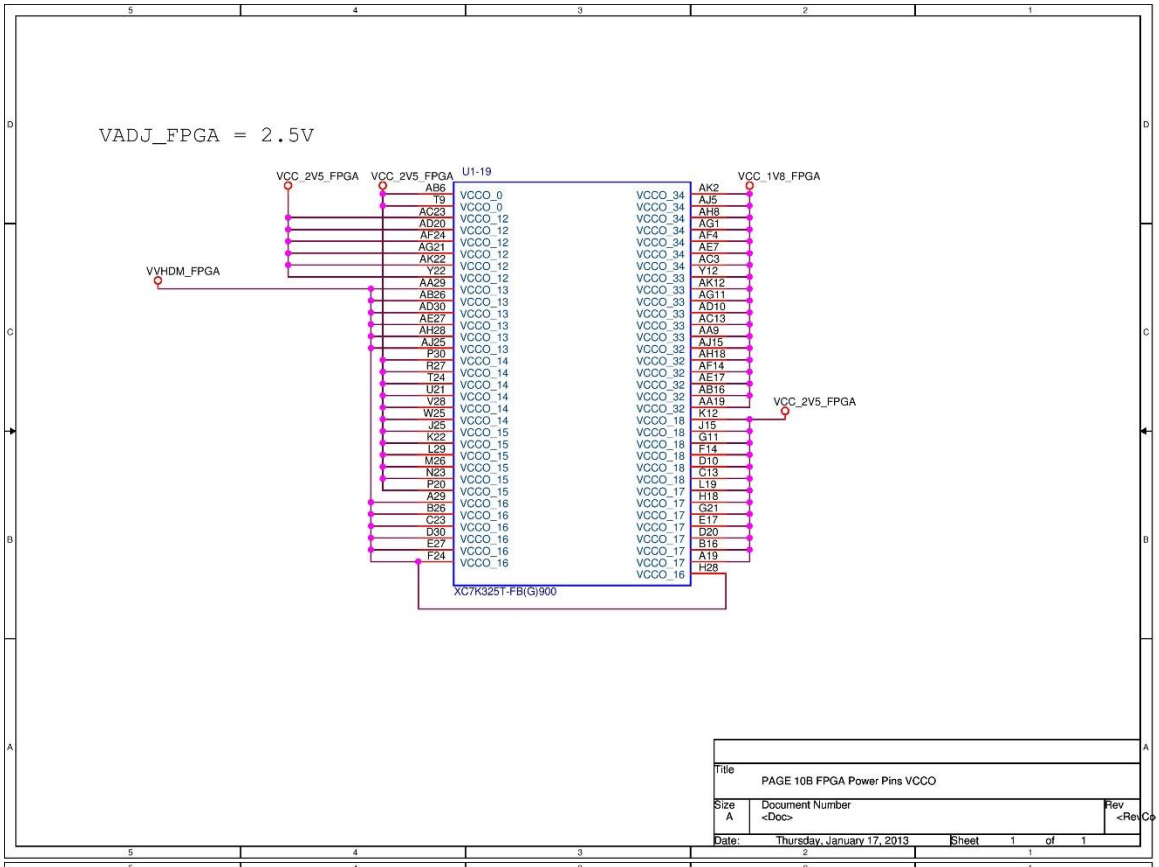

T023 not used
U1-7
BANK_17
XC7K325T-FB(G)900
PE_CONN_D_IO43
PE_CONN_D_IO33
PE_CONN_D_IO51
PE_CONN_D_IO25
PE_CONN_D_IO29
PE_CONN_D_IO50
PE_CONN_D_IO31
PE_CONN_D_IO30
PE_CONN_D_IO46
PE_CONN_D_IO35
PE_CONN_D_IO26
PE_CONN_D_IO54
PE_CONN_D_IO38
PE_CONN_D_IO4
PE_CONN_D_IO28
PE_CONN_D_IO58
PE_CONN_D_IO53
PE_CONN_D_IO55
PE_CONN_D_IO27
PE_CONN_D_IO47
PE_CONN_D_IO49
PE_CONN_D_IO24
PE_CONN_D_IO52
PE_CONN_D_IO40
PE_CONN_D_IO45
PE_CONN_D_IO59
PE_CONN_D_IO57
PE_CONN_D_IO48
PE_CONN_D_IO32
PE_CONN_D_IO44
PE_CONN_D_IO42
PE_CONN_D_IO30
PE_CONN_D_IO28
PE_CONN_D_IO34
PE_CONN_D_IO38
PE_CONN_D_IO56
VCC_2V5_FPGA
R126
?/0402
R127
?/0402
R128
?/0402
R129
?/0402
Not used
T023 not used
U1-8
BANK_18
XC7K325T-FB(G)900
PE_CONN_C_IO51
PE_CONN_C_IO33
PE_CONN_C_IO35
PE_CONN_C_IO45
PE_CONN_C_IO37
PE_CONN_C_IO53
PE_CONN_C_IO43
PE_CONN_C_IO4
PE_CONN_C_IO49
PE_CONN_C_IO26
PE_CONN_C_IO55
PE_CONN_C_IO59
PE_CONN_C_IO47
PE_CONN_C_IO28
PE_CONN_C_IO24
PE_CONN_C_IO29
PE_CONN_C_IO57
PE_CONN_C_IO58
PE_CONN_C_IO52
PE_CONN_C_IO27
PE_CONN_C_IO31
PE_CONN_C_IO34
PE_CONN_C_IO42
PE_CONN_C_IO36
PE_CONN_C_IO30
PE_CONN_C_IO32
PE_CONN_C_IO54
PE_CONN_C_IO40
PE_CONN_C_IO56
PE_CONN_C_IO50
PE_CONN_C_IO48
PE_CONN_C_IO33
PE_CONN_C_IO46
PE_CONN_C_IO60
PE_CONN_C_IO25
PE_CONN_C_IO38
VCC_2V5_FPGA
R168
?/0402
R169
?/0402
R130
?/0402
R131
?/0402
Not used
Title
PAGE 06 FPGA Banks 17 18
Size
A
Document Number
<Doc>
Rev
<Rev.Co
Date:
Thursday, January 10, 2013
Sheet
1
of
1

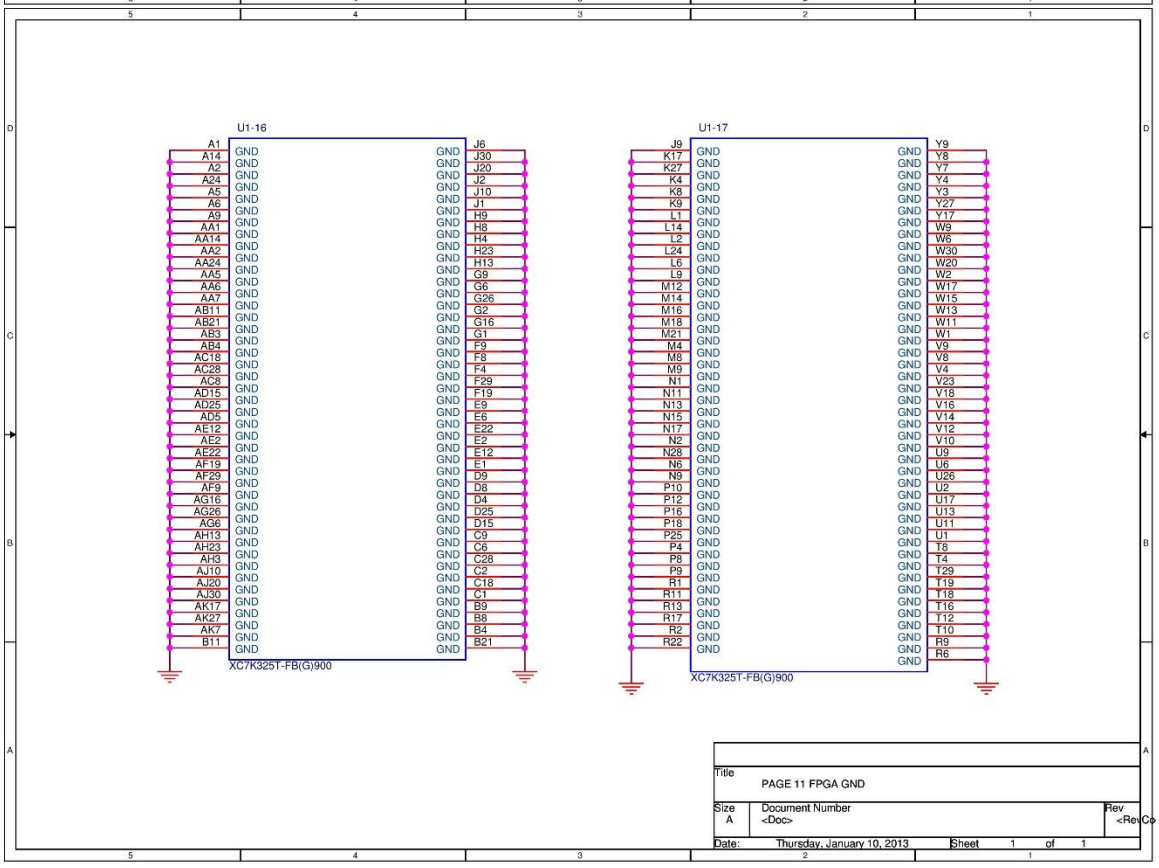

1023 not used
U1-9
BANK_32
XC7K325T-FB(G)900
VCC_1V8_FPGA
R134 ?/0402
R135 ?/0402
R132 ?/0402
R133 ?/0402
Not used
U1-10
BANK_33
XC7K325T-FB(G)900
SYSCLK_N
SYSCLK_P
R204 ?/0402
R205 ?/0402
R206 ?/0402
R207 ?/0402
Title
PAGE 07 FPGA Banks 32 33
Size A
Document Number <Doc>
Rev <RevCo
Date: Friday, January 18, 2013
Sheet 1 of 1

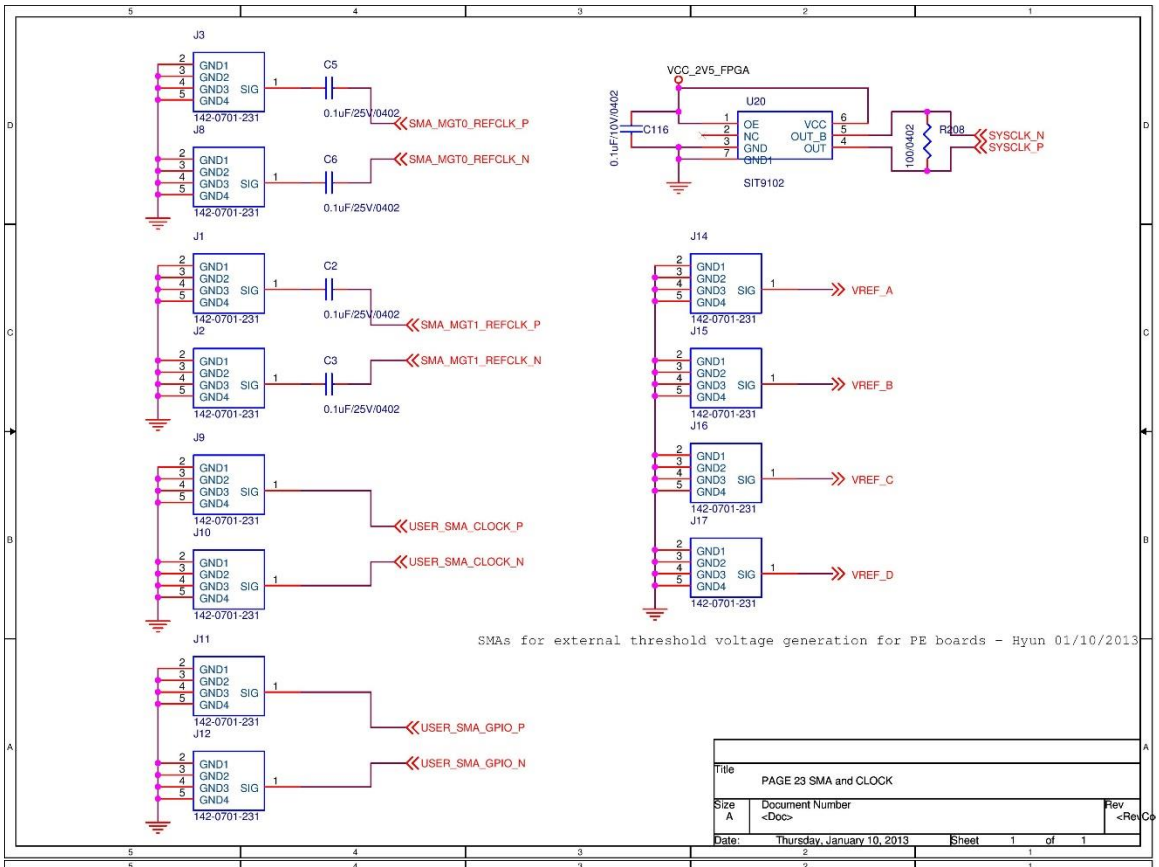

Shadow Sampler Data Acquisition
U1-11
BANK_34
XC7K325T-FB(G)900
VCC_1V8_FPGA
R170 ?/0402
R171 ?/0402
R172 ?/0402
R173 ?/0402
PE_CONN_C_SS1_N
PE_CONN_C_SS1_P
PE_CONN_C_SS0_N
PE_CONN_C_SS0_P
PE_CONN_B_SS2_N
PE_CONN_B_SS2_P
PE_CONN_B_SS0_N
PE_CONN_B_SS0_P
PE_CONN_B_SS3_N
PE_CONN_B_SS3_P
PE_CONN_A_SS1_N
PE_CONN_A_SS1_P
PE_CONN_A_SS3_N
PE_CONN_A_SS3_P
PE_CONN_A_SS2_N
PE_CONN_A_SS2_P
PE_CONN_D_SS0_N
PE_CONN_D_SS0_P
PE_CONN_B_SS1_P
PE_CONN_B_SS1_N
PE_CONN_C_SS3_P
PE_CONN_C_SS3_N
PE_CONN_C_SS2_P
PE_CONN_C_SS2_N
PE_CONN_A_SS_CLK_P
PE_CONN_A_SS_CLK_N
PE_CONN_D_SS1_P
PE_CONN_D_SS1_N
PE_CONN_D_SS2_P
PE_CONN_D_SS2_N
PE_CONN_D_SS3_P
PE_CONN_D_SS3_N
PE_CONN_A_SS0_P
PE_CONN_A_SS0_N
PE_CONN_B_SS_CLK_P
PE_CONN_B_SS_CLK_N
PE_CONN_C_SS_CLK_P
PE_CONN_C_SS_CLK_N
PE_CONN_D_SS_CLK_P
PE_CONN_D_SS_CLK_N
Clk distributed to PE Connectors A-D using a buffer?
Title
PAGE 08 FPGA Bank 34
Size A
Document Number <Doc>
Rev <RevCo
Date: Thursday, January 17, 2013
Sheet 1 of 1

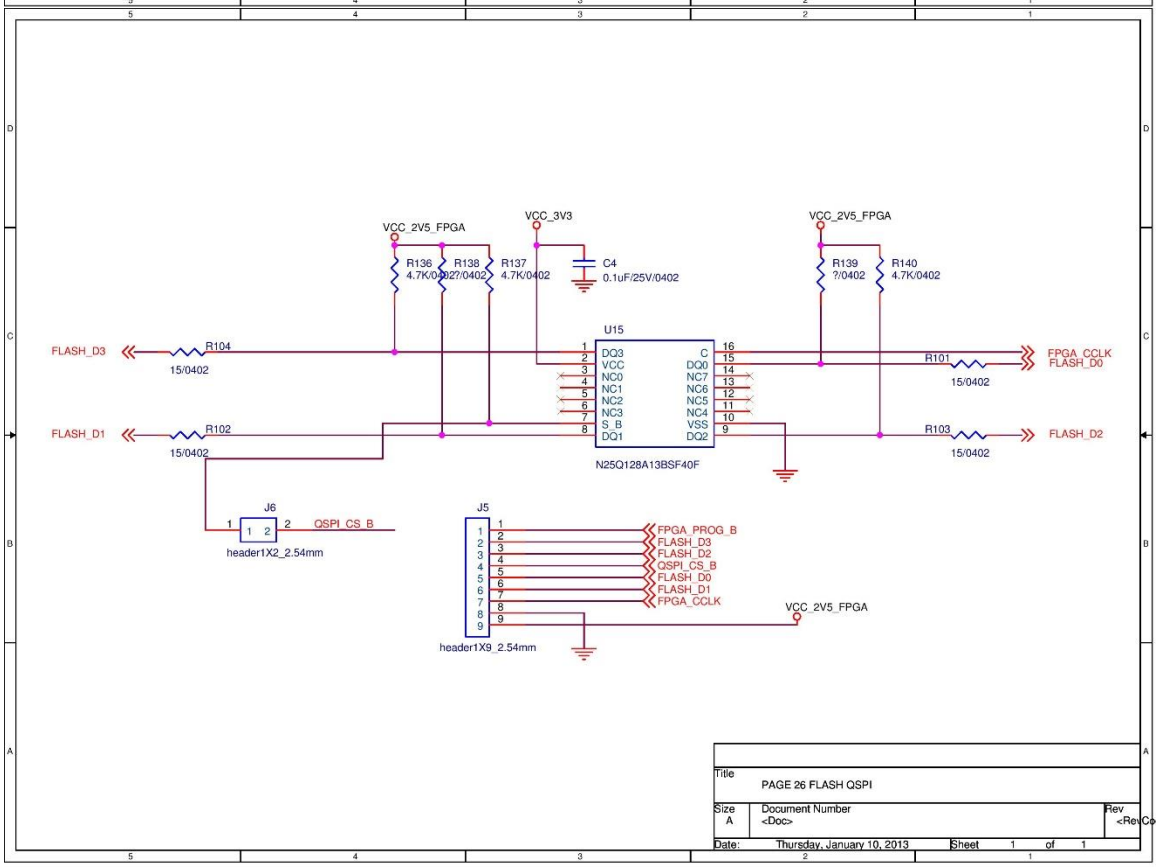

VCC_1V2_FPGA
U1-12
R141
100/1%/1/2
SMA_MGT0_REFCLK_N
SMA_MGT0_REFCLK_P
MGT_A0_RX_N
MGT_A1_RX_N
MGT_A2_RX_N
MGT_A3_RX_N
MGT_A0_RX_P
MGTAVTTRCAL_115
MGTREFCLK0N_115
MGTREFCLK0P_115
MGTREFCLK1N_115
MGTREFCLK1P_115
MGTRREF_115
MGTXRXN0_115
MGTXRXN1_115
MGTXRXN2_115
MGTXRXN3_115
MGTXRXP0_115
MGTXTXP3_115
MGTXTXP2_115
MGTXTXP1_115
MGTXTXP0_115
MGTXTXN3_115
MGTXTXN2_115
MGTXTXN1_115
MGTXTXN0_115
MGTXRXP3_115
MGTXRXP2_115
MGTXRXP1_115
MGT_A3_TX_P
MGT_A2_TX_P
MGT_A1_TX_P
MGT_A0_TX_P
MGT_A3_TX_N
MGT_A2_TX_N
MGT_A1_TX_N
MGT_A0_TX_N
MGT_A3_RX_P
MGT_A2_RX_P
MGT_A1_RX_P
XC7K325T-FB(G)900
U1-13
MGT_B0_RX_N
MGT_B1_RX_N
MGT_B2_RX_N
MGT_B3_RX_N
MGT_B0_RX_P
MGT_B1_RX_P
MGTREFCLK0N_116
MGTREFCLK0P_116
MGTREFCLK1N_116
MGTREFCLK1P_116
MGTXRXN0_116
MGTXRXN1_116
MGTXRXN2_116
MGTXRXN3_116
MGTXRXP0_116
MGTXRXP1_116
MGTXTXP3_116
MGTXTXP2_116
MGTXTXP1_116
MGTXTXP0_116
MGTXTXN3_116
MGTXTXN2_116
MGTXTXN1_116
MGTXTXN0_116
MGTXRXP3_116
MGTXRXP2_116
MGT_B3_TX_P
MGT_B2_TX_P
MGT_B1_TX_P
MGT_B0_TX_P
MGT_B3_TX_N
MGT_B2_TX_N
MGT_B1_TX_N
MGT_B0_TX_N
MGT_B3_RX_P
MGT_B2_RX_P
XC7K325T-FB(G)900
U1-14
SMA_MGT1_REFCLK_N
SMA_MGT1_REFCLK_P
MGT_C0_RX_N
MGT_C1_RX_N
MGT_C2_RX_N
MGT_C3_RX_N
MGT_C0_RX_P
MGT_C1_RX_P
MGTREFCLK0N_117
MGTREFCLK0P_117
MGTREFCLK1N_117
MGTREFCLK1P_117
MGTXRXN0_117
MGTXRXN1_117
MGTXRXN2_117
MGTXRXN3_117
MGTXRXP0_117
MGTXRXP1_117
MGTXTXP3_117
MGTXTXP2_117
MGTXTXP1_117
MGTXTXP0_117
MGTXTXN3_117
MGTXTXN2_117
MGTXTXN1_117
MGTXTXN0_117
MGTXRXP3_117
MGTXRXP2_117
MGT_C3_TX_P
MGT_C2_TX_P
MGT_C1_TX_P
MGT_C0_TX_P
MGT_C3_TX_N
MGT_C2_TX_N
MGT_C1_TX_N
MGT_C0_TX_N
MGT_C3_RX_P
MGT_C2_RX_P
XC7K325T-FB(G)900
U1-15
MGT_D0_RX_N
MGT_D1_RX_N
MGT_D2_RX_N
MGT_D3_RX_N
MGT_D0_RX_P
MGT_D1_RX_P
MGTREFCLK0N_118
MGTREFCLK0P_118
MGTREFCLK1N_118
MGTREFCLK1P_118
MGTXRXN0_118
MGTXRXN1_118
MGTXRXN2_118
MGTXRXN3_118
MGTXRXP0_118
MGTXRXP1_118
MGTXTXP3_118
MGTXTXP2_118
MGTXTXP1_118
MGTXTXP0_118
MGTXTXN3_118
MGTXTXN2_118
MGTXTXN1_118
MGTXTXN0_118
MGTXRXP3_118
MGTXRXP2_118
MGT_D3_TX_P
MGT_D2_TX_P
MGT_D1_TX_P
MGT_D0_TX_P
MGT_D3_TX_N
MGT_D2_TX_N
MGT_D1_TX_N
MGT_D0_TX_N
MGT_D3_RX_P
MGT_D2_RX_P
XC7K325T-FB(G)900
Title
PAGE 09 FPGA GT Banks
Size
A
Document Number
<Doc>
Rev
<RevCo
Date:
Thursday, January 10, 2013
Sheet
1
of
1

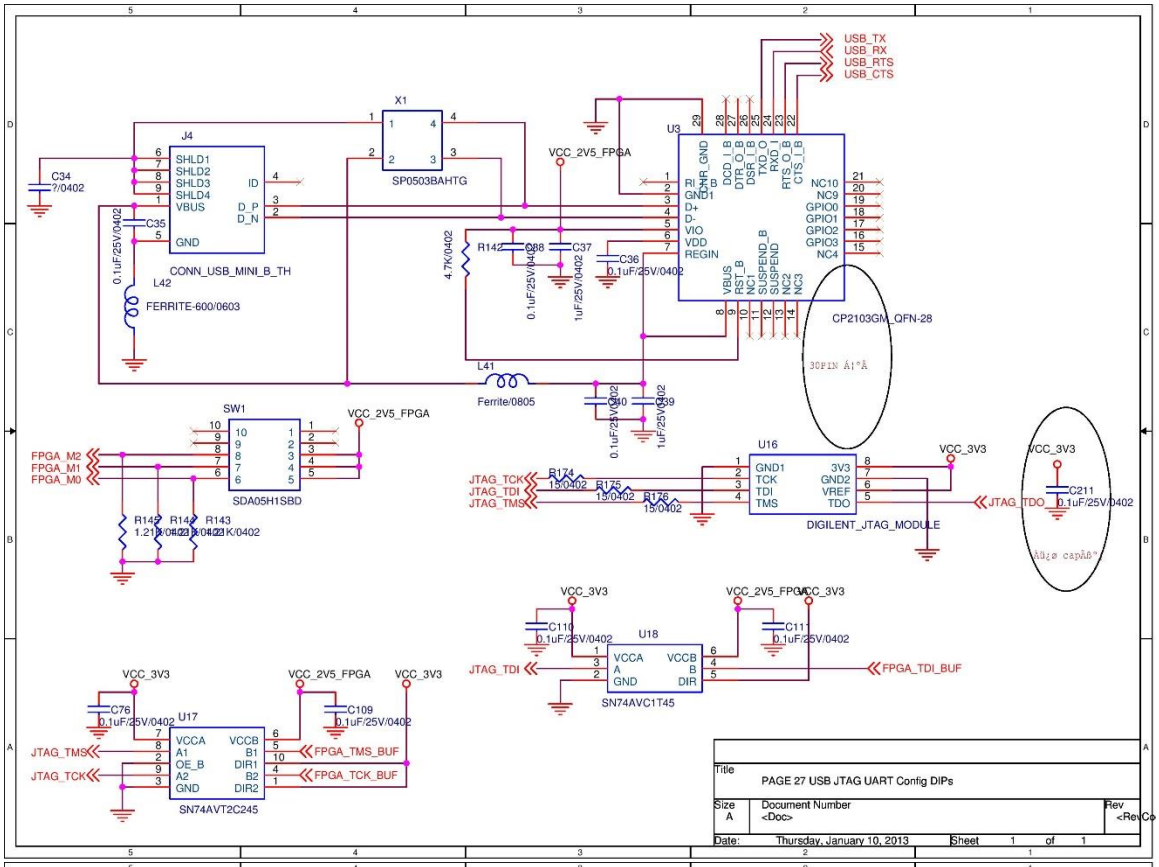

MGTAVCC = 1.0V
VCCINT_FPGA = 1.0V
VCC_1V_FPGA
U1-18
MGTAVTT = 1.2V
VCC_1V2_FPGA
MGTVCCAUX = 1.8V
VCC_1V8_FPGA
VCCAUX = 1.8V
VCCBRAM = 1.0V
VCCAUX_IO = 1.8V
MGTAVCC
MGTAVTT
MGTVCCAUX
VCCAUX
VCCINT
VCCBRAM
VCCAUX_IO_G0
XC7K325T-FB(G)900
Title
PAGE 10A FPGA Power Pins
Size
A
Document Number
<Doc>
Rev
<RevCo
Date:
Thursday, January 10, 2013
Sheet
1
of
1

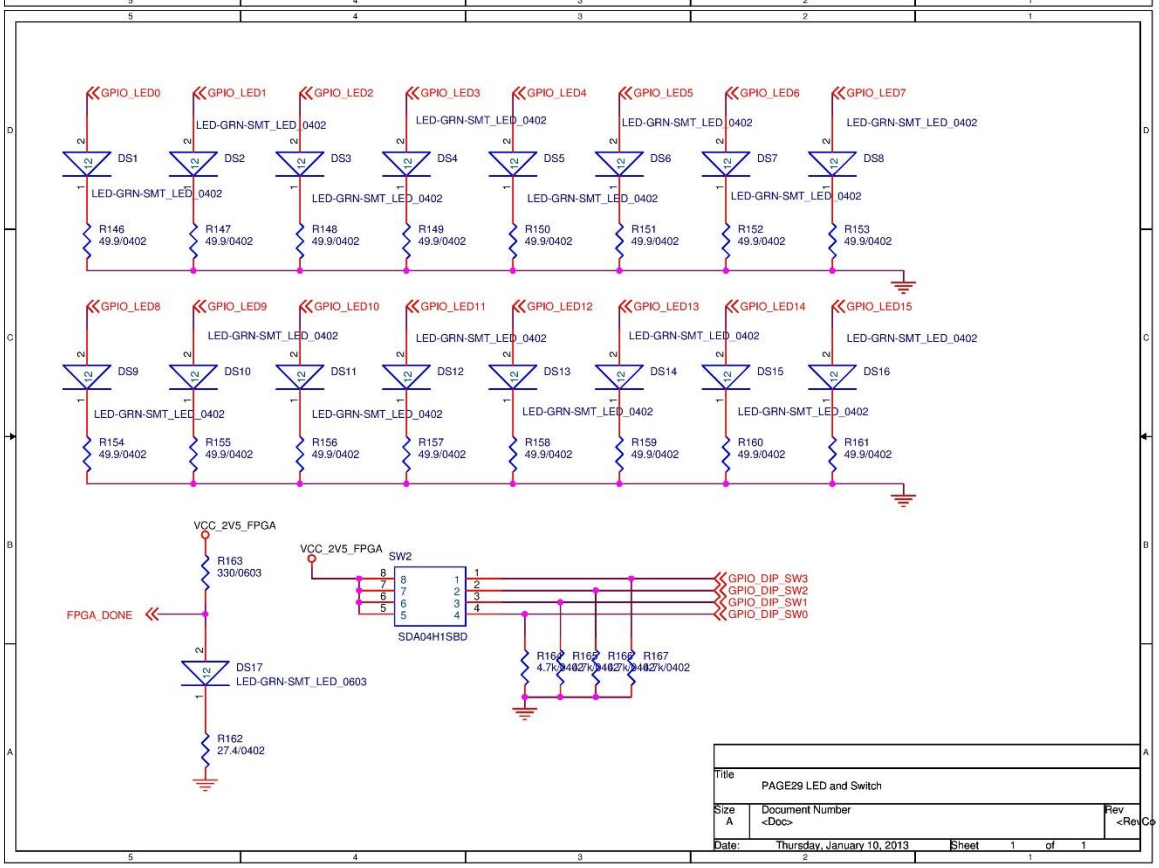

VADJ_FPGA = 2.5V
VCC_2V5_FPGA
VCC_2V5_FPGA
U1-19
VCC_1V8_FPGA
VVHDM_FPGA
VCC_2V5_FPGA
XC7K325T-FBG900
Title
PAGE 10B FPGA Power Pins VCCO
Size
A
Document Number
<Doc>
Rev
<RevCo
Date:
Thursday, January 17, 2013
Sheet
1
of
1

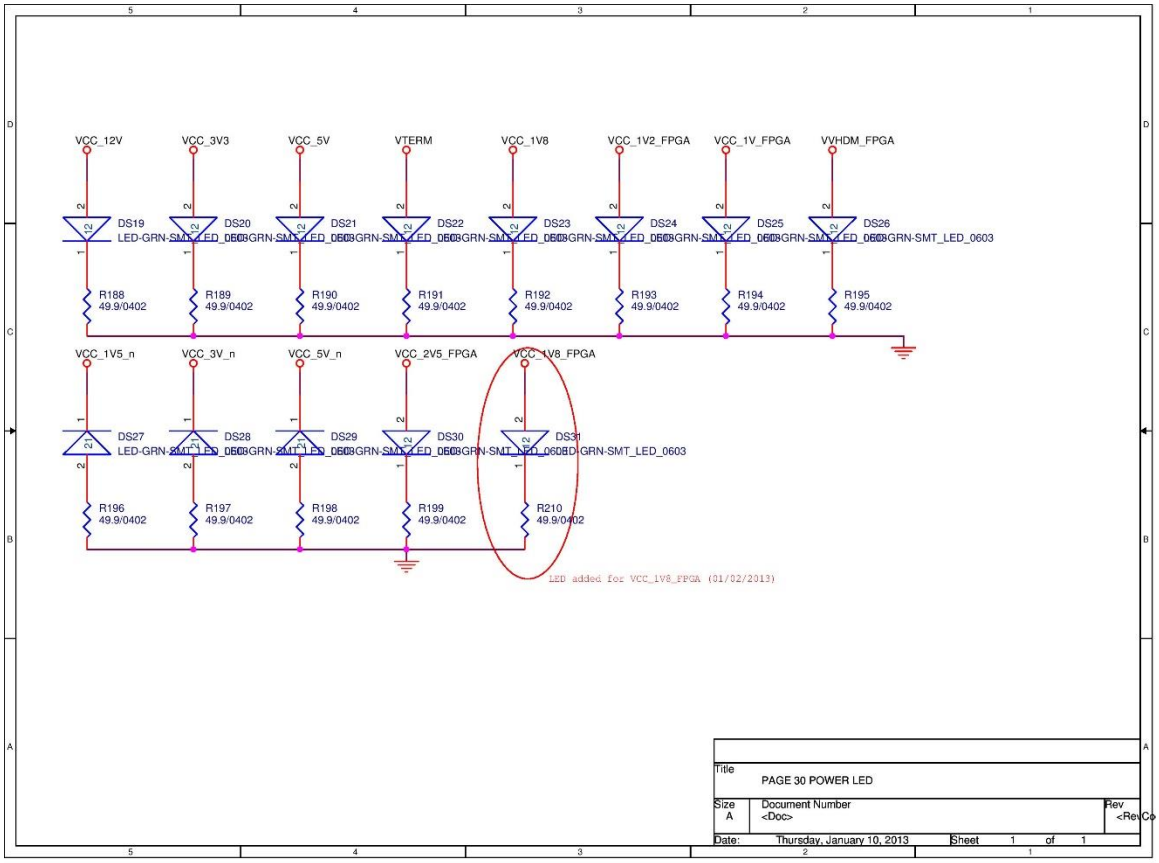

U1-16
U1-17
XC7K325T-FBG900
XC7K325T-FBG900
Title
PAGE 11 FPGA GND
Size
A
Document Number
<Doc>
Rev
<RevCo
Date:
Thursday, January 10, 2013
Sheet
1
of
1

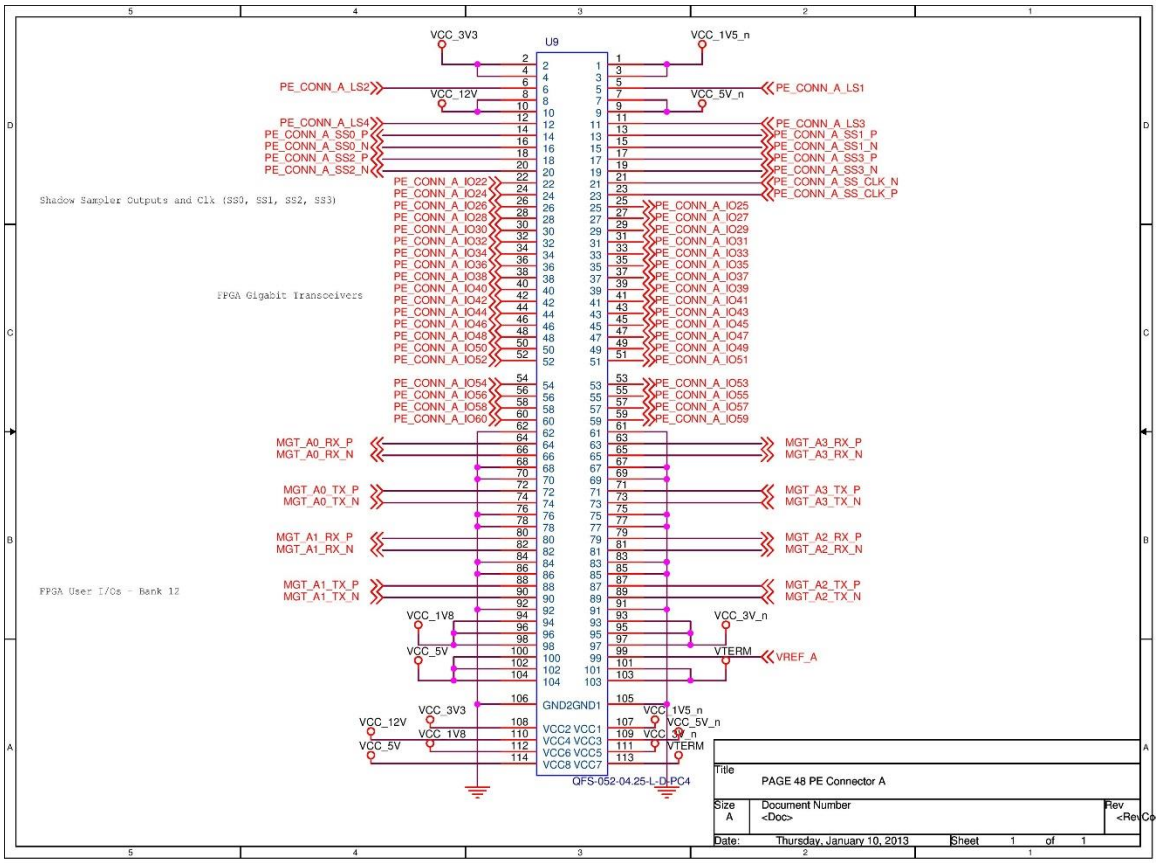

J3
J8
J1
J2
J9
J10
J11
J12
J14
J15
J16
J17
142-0701-231
C5
C6
C2
C3
0.1uF/25V/0402
SMA_MGT0_REFCLK_P
SMA_MGT0_REFCLK_N
SMA_MGT1_REFCLK_P
SMA_MGT1_REFCLK_N
USER_SMA_CLOCK_P
USER_SMA_CLOCK_N
USER_SMA_GPIO_P
USER_SMA_GPIO_N
VREF_A
VREF_B
VREF_C
VREF_D
VCC_2V5_FPGA
U20
C116
0.1uF/10V/0402
SIT9102
R208
100/0402
SYSCLK_N
SYSCLK_P
SMAs for external threshold voltage generation for PE boards - Hyun 01/10/2013
Title
PAGE 23 SMA and CLOCK
Size
A
Document Number
<Doc>
Rev
<Rev.Co
Date:
Thursday, January 10, 2013
Sheet
1
of
1

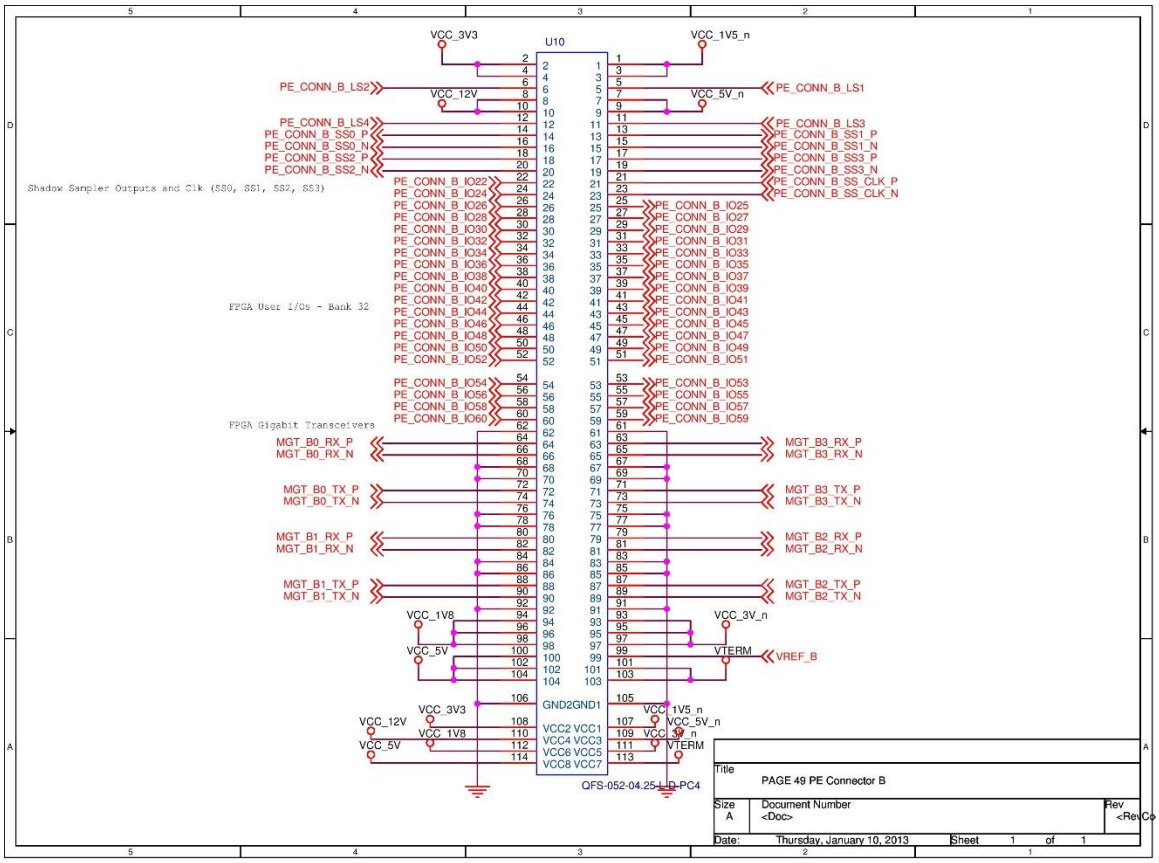

VCC_2V5_FPGA
VCC_3V3
R136
4.7K/0402
R138
?/0402
R137
4.7K/0402
C4
0.1uF/25V/0402
R139
?/0402
R140
4.7K/0402
U15
FLASH_D3
R104
15/0402
FLASH_D1
R102
15/0402
R101
15/0402
R103
15/0402
FPGA_CCLK
FLASH_D0
FLASH_D2
DQ3
VCC
NC0
NC1
NC2
NC3
S_B
DQ1
C
DQ0
NC7
NC6
NC5
NC4
VSS
DQ2
N25Q128A13BSF40F
J6
QSPI_CS_B
header1X2_2.54mm
J5
FPGA_PROG_B
FLASH_D3
FLASH_D2
QSPI_CS_B
FLASH_D0
FLASH_D1
FPGA_CCLK
header1X9_2.54mm
Title
PAGE 26 FLASH QSPI
Size
A
Document Number
<Doc>
Rev
<Rev.Co
Date:
Thursday, January 10, 2013
Sheet
1
of
1

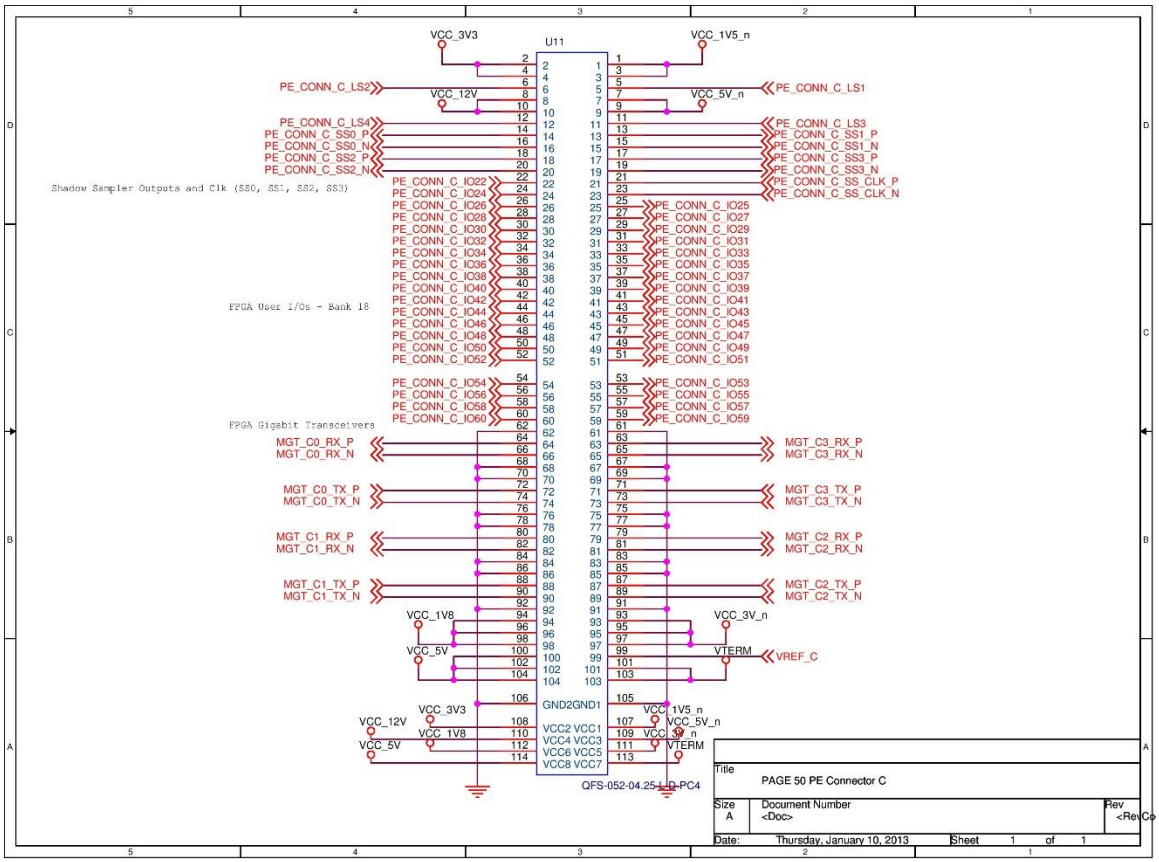

USB_TX
USB_RX
USB_RTS
USB_CTS
X1
SP0503BAHTG
J4
CONN_USB_MINI_B_TH
FERRITE-600/0603
L42
L41
Ferrite/0805
U3
CP2103GM_QFN-28
VCC_2V5_FPGA
SW1
SDA05H1SBD
FPGA_M2
FPGA_M1
FPGA_M0
U16
DIGILENT_JTAG_MODULE
JTAG_TCK
JTAG_TDI
JTAG_TMS
JTAG_TDO
VCC_3V3
U18
SN74AVC1T45
FPGA_TDI_BUF
U17
SN74AVT2C245
FPGA_TMS_BUF
FPGA_TCK_BUF
Title
PAGE 27 USB JTAG UART Config DIPs
Size
A
Document Number
<Doc>
Rev
Date:
Thursday, January 10, 2013
Sheet
1
of
1

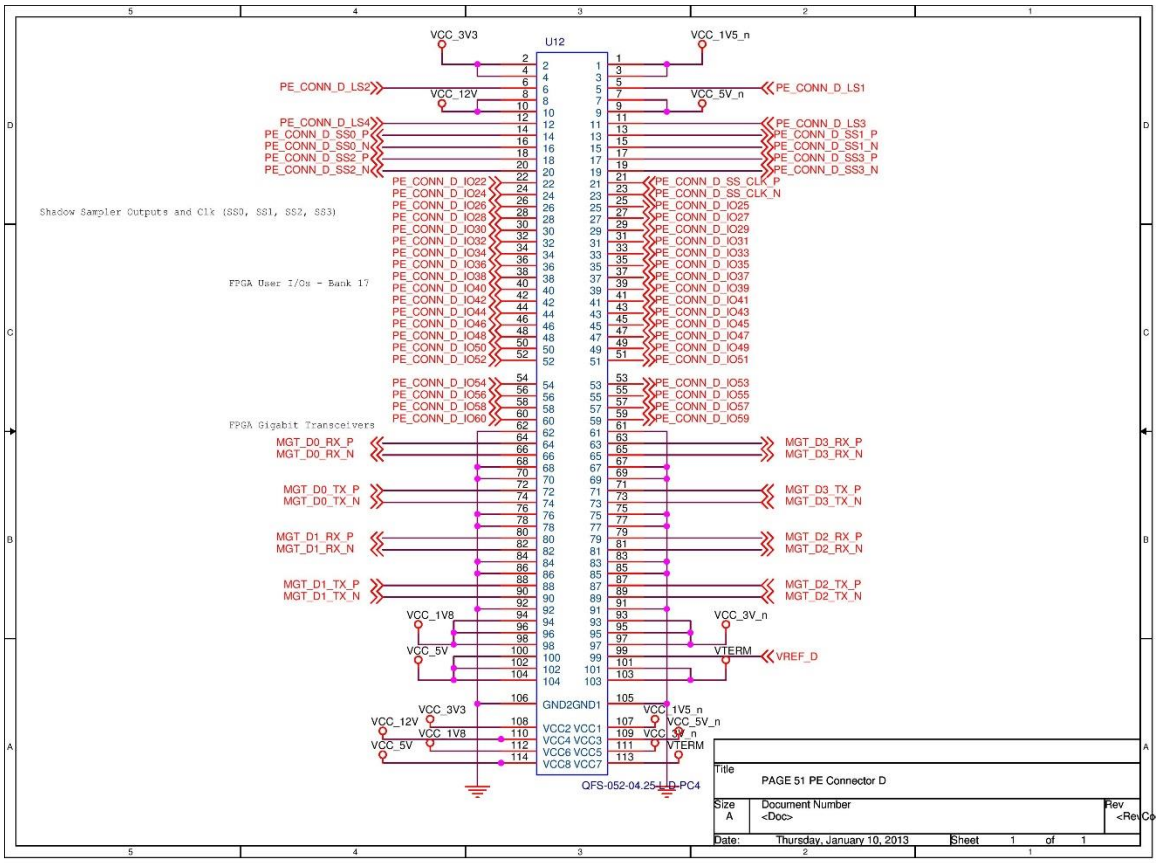

GPIO_LED0
GPIO_LED1
GPIO_LED2
GPIO_LED3
GPIO_LED4
GPIO_LED5
GPIO_LED6
GPIO_LED7
GPIO_LED8
GPIO_LED9
GPIO_LED10
GPIO_LED11
GPIO_LED12
GPIO_LED13
GPIO_LED14
GPIO_LED15
LED-GRN-SMT_LED_0402
LED-GRN-SMT_LED_0603
VCC_2V5_FPGA
FPGA_DONE
SW2
SDA04H1SBD
GPIO_DIP_SW3
GPIO_DIP_SW2
GPIO_DIP_SW1
GPIO_DIP_SW0
Title
PAGE29 LED and Switch
Size
A
Document Number
<Doc>
Rev
Date:
Thursday, January 10, 2013
Sheet
1
of
1

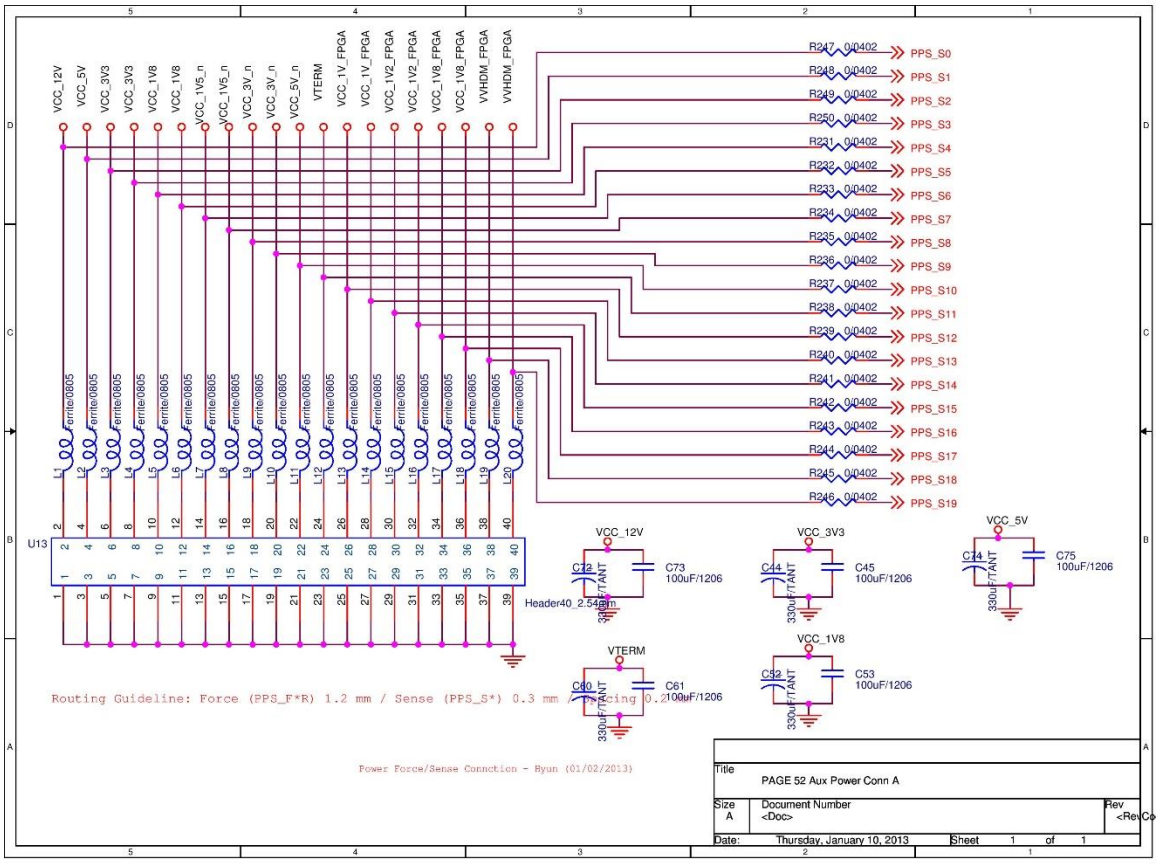

VCC_12V
VCC_3V3
VCC_5V
VTERM
VCC_1V8
VCC_1V2_FPGA
VCC_1V_FPGA
VVHDM_FPGA
DS19 LED-GRN-SMT_LED_0603
DS20 LED-GRN-SMT_LED_0603
DS21 LED-GRN-SMT_LED_0603
DS22 LED-GRN-SMT_LED_0603
DS23 LED-GRN-SMT_LED_0603
DS24 LED-GRN-SMT_LED_0603
DS25 LED-GRN-SMT_LED_0603
DS26 LED-GRN-SMT_LED_0603
R188 49.9/0402
R189 49.9/0402
R190 49.9/0402
R191 49.9/0402
R192 49.9/0402
R193 49.9/0402
R194 49.9/0402
R195 49.9/0402
VCC_1V5_n
VCC_3V_n
VCC_5V_n
VCC_2V5_FPGA
VCC_1V8_FPGA
DS27 LED-GRN-SMT_LED_0603
DS28 LED-GRN-SMT_LED_0603
DS29 LED-GRN-SMT_LED_0603
DS30 LED-GRN-SMT_LED_0603
DS31 LED-GRN-SMT_LED_0603
R196 49.9/0402
R197 49.9/0402
R198 49.9/0402
R199 49.9/0402
R210 49.9/0402
LED added for VCC_1V8_FPGA (01/02/2013)
Title
PAGE 30 POWER LED
Size A
Document Number <Doc>
Rev <Rev
Date: Thursday, January 10, 2013
Sheet 1 of 1

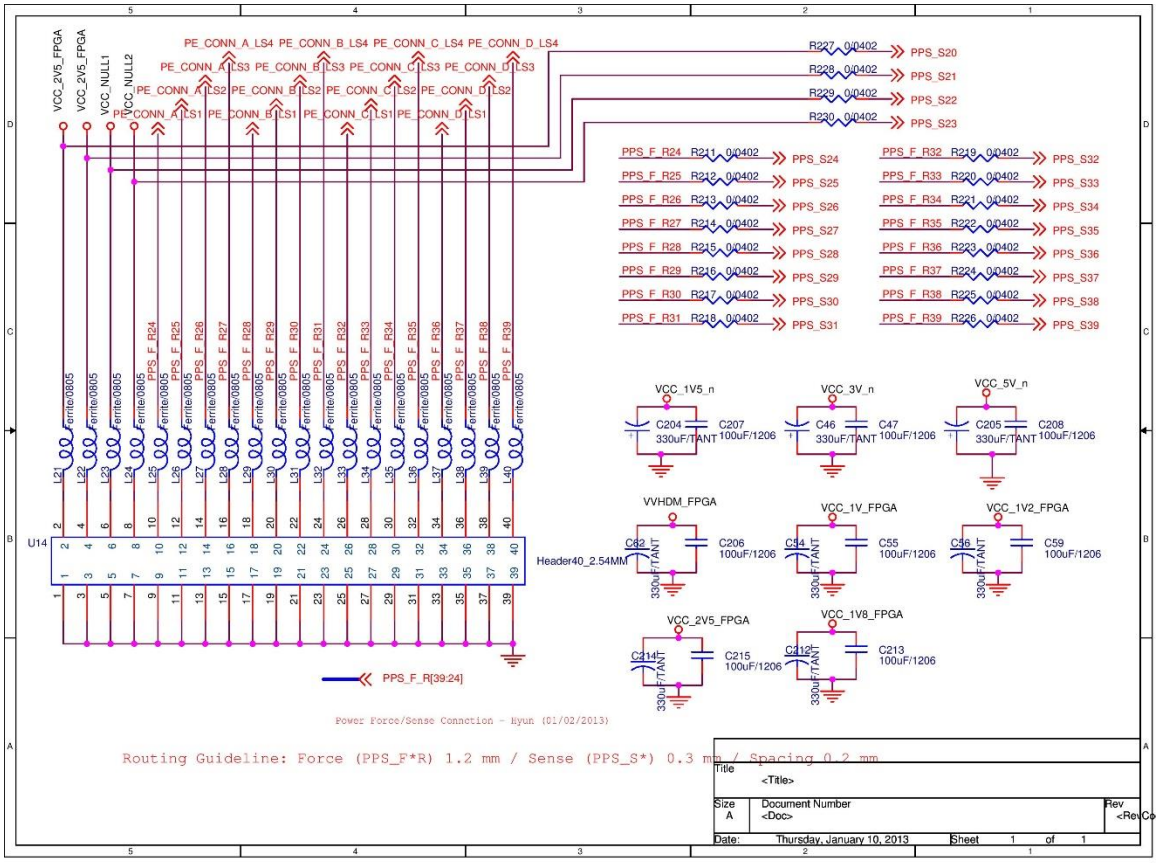

U9
VCC_3V3
VCC_1V5_n
PE_CONN_A_LS2
VCC_12V
VCC_5V_n
PE_CONN_A_LS1
PE_CONN_A_LS4
PE_CONN_A_SS0_P
PE_CONN_A_SS0_N
PE_CONN_A_SS2_P
PE_CONN_A_SS2_N
PE_CONN_A_LS3
PE_CONN_A_SS1_P
PE_CONN_A_SS1_N
PE_CONN_A_SS3_P
PE_CONN_A_SS3_N
PE_CONN_A_SS_CLK_N
PE_CONN_A_SS_CLK_P
Shadow Sampler Outputs and Clk (SS0, SS1, SS2, SS3)
PE_CONN_A_IO22
PE_CONN_A_IO24
PE_CONN_A_IO26
PE_CONN_A_IO28
PE_CONN_A_IO30
PE_CONN_A_IO32
PE_CONN_A_IO34
PE_CONN_A_IO36
PE_CONN_A_IO38
PE_CONN_A_IO40
PE_CONN_A_IO42
PE_CONN_A_IO44
PE_CONN_A_IO46
PE_CONN_A_IO48
PE_CONN_A_IO50
PE_CONN_A_IO52
PE_CONN_A_IO25
PE_CONN_A_IO27
PE_CONN_A_IO29
PE_CONN_A_IO31
PE_CONN_A_IO33
PE_CONN_A_IO35
PE_CONN_A_IO37
PE_CONN_A_IO39
PE_CONN_A_IO41
PE_CONN_A_IO43
PE_CONN_A_IO45
PE_CONN_A_IO47
PE_CONN_A_IO49
PE_CONN_A_IO51
FPGA Gigabit Transceivers
PE_CONN_A_IO54
PE_CONN_A_IO56
PE_CONN_A_IO58
PE_CONN_A_IO60
PE_CONN_A_IO53
PE_CONN_A_IO55
PE_CONN_A_IO57
PE_CONN_A_IO59
MGT_A0_RX_P
MGT_A0_RX_N
MGT_A3_RX_P
MGT_A3_RX_N
MGT_A0_TX_P
MGT_A0_TX_N
MGT_A3_TX_P
MGT_A3_TX_N
MGT_A1_RX_P
MGT_A1_RX_N
MGT_A2_RX_P
MGT_A2_RX_N
MGT_A1_TX_P
MGT_A1_TX_N
MGT_A2_TX_P
MGT_A2_TX_N
FPGA User I/Os - Bank 12
VCC_1V8
VCC_3V_n
VCC_5V
VTERM
VREF_A
GND2GND1
VCC2 VCC1
VCC4 VCC3
VCC6 VCC5
VCC8 VCC7
QFS-052-04.25-L-D-PC4
Title
PAGE 48 PE Connector A
Size A
Document Number <Doc>
Rev <Rev
Date: Thursday, January 10, 2013
Sheet 1 of 1

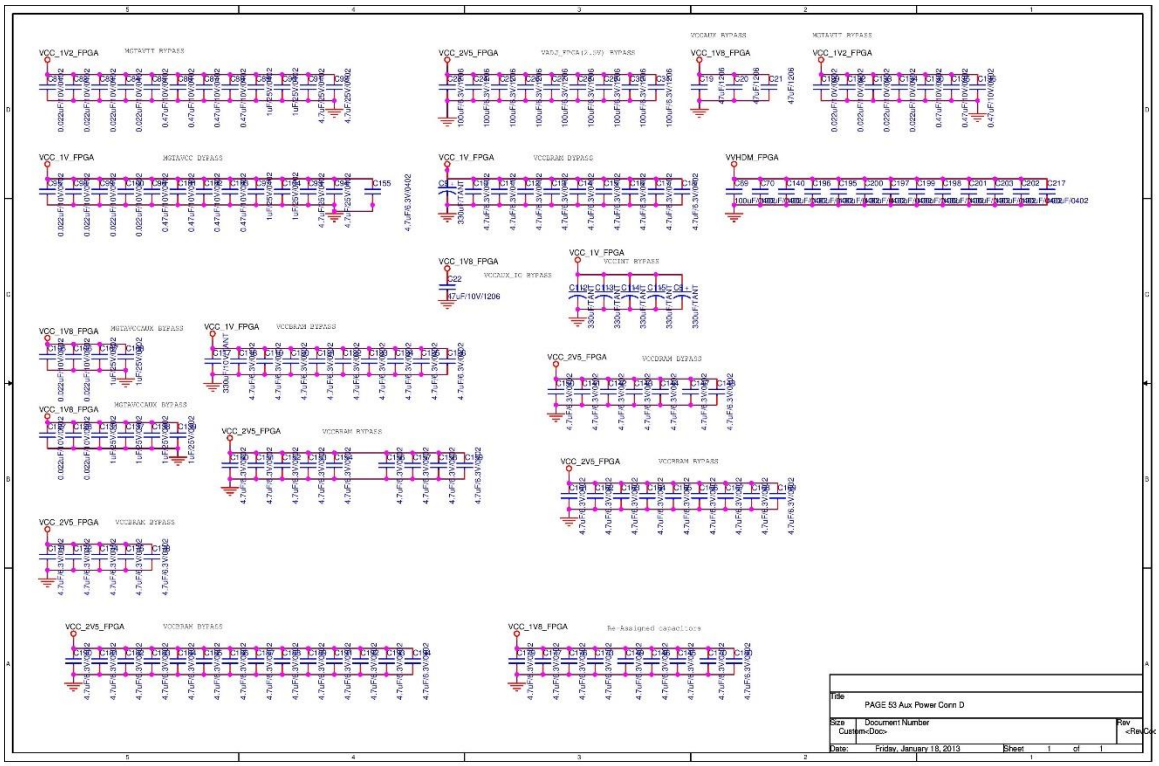

U10
VCC_3V3
VCC_1V5_n
VCC_12V
VCC_5V_n
PE_CONN_B_LS2
PE_CONN_B_LS1
PE_CONN_B_LS4
PE_CONN_B_LS3
PE_CONN_B_SS0_P
PE_CONN_B_SS1_P
PE_CONN_B_SS0_N
PE_CONN_B_SS1_N
PE_CONN_B_SS2_P
PE_CONN_B_SS3_P
PE_CONN_B_SS2_N
PE_CONN_B_SS3_N
PE_CONN_B_SS_CLK_P
PE_CONN_B_SS_CLK_N
Shadow Sampler Outputs and Clk (SS0, SS1, SS2, SS3)
PE_CONN_B_IO22
PE_CONN_B_IO24
PE_CONN_B_IO26
PE_CONN_B_IO28
PE_CONN_B_IO30
PE_CONN_B_IO32
PE_CONN_B_IO34
PE_CONN_B_IO36
PE_CONN_B_IO38
PE_CONN_B_IO40
PE_CONN_B_IO42
PE_CONN_B_IO44
PE_CONN_B_IO46
PE_CONN_B_IO48
PE_CONN_B_IO50
PE_CONN_B_IO52
PE_CONN_B_IO25
PE_CONN_B_IO27
PE_CONN_B_IO29
PE_CONN_B_IO31
PE_CONN_B_IO33
PE_CONN_B_IO35
PE_CONN_B_IO37
PE_CONN_B_IO39
PE_CONN_B_IO41
PE_CONN_B_IO43
PE_CONN_B_IO45
PE_CONN_B_IO47
PE_CONN_B_IO49
PE_CONN_B_IO51
FPGA User I/Os - Bank 32
PE_CONN_B_IO54
PE_CONN_B_IO56
PE_CONN_B_IO58
PE_CONN_B_IO60
PE_CONN_B_IO53
PE_CONN_B_IO55
PE_CONN_B_IO57
PE_CONN_B_IO59
FPGA Gigabit Transceivers
MGT_B0_RX_P
MGT_B0_RX_N
MGT_B3_RX_P
MGT_B3_RX_N
MGT_B0_TX_P
MGT_B0_TX_N
MGT_B3_TX_P
MGT_B3_TX_N
MGT_B1_RX_P
MGT_B1_RX_N
MGT_B2_RX_P
MGT_B2_RX_N
MGT_B1_TX_P
MGT_B1_TX_N
MGT_B2_TX_P
MGT_B2_TX_N
VCC_1V8
VCC_3V_n
VCC_5V
VTERM
VREF_B
GND2GND1
VCC2 VCC1
VCC4 VCC3
VCC6 VCC5
VCC8 VCC7
QFS-052-04.25-L-D-PC4
Title
PAGE 49 PE Connector B
Size
A
Document Number
<Doc>
Rev
<RevCo
Date:
Thursday, January 10, 2013
Sheet
1
of
1

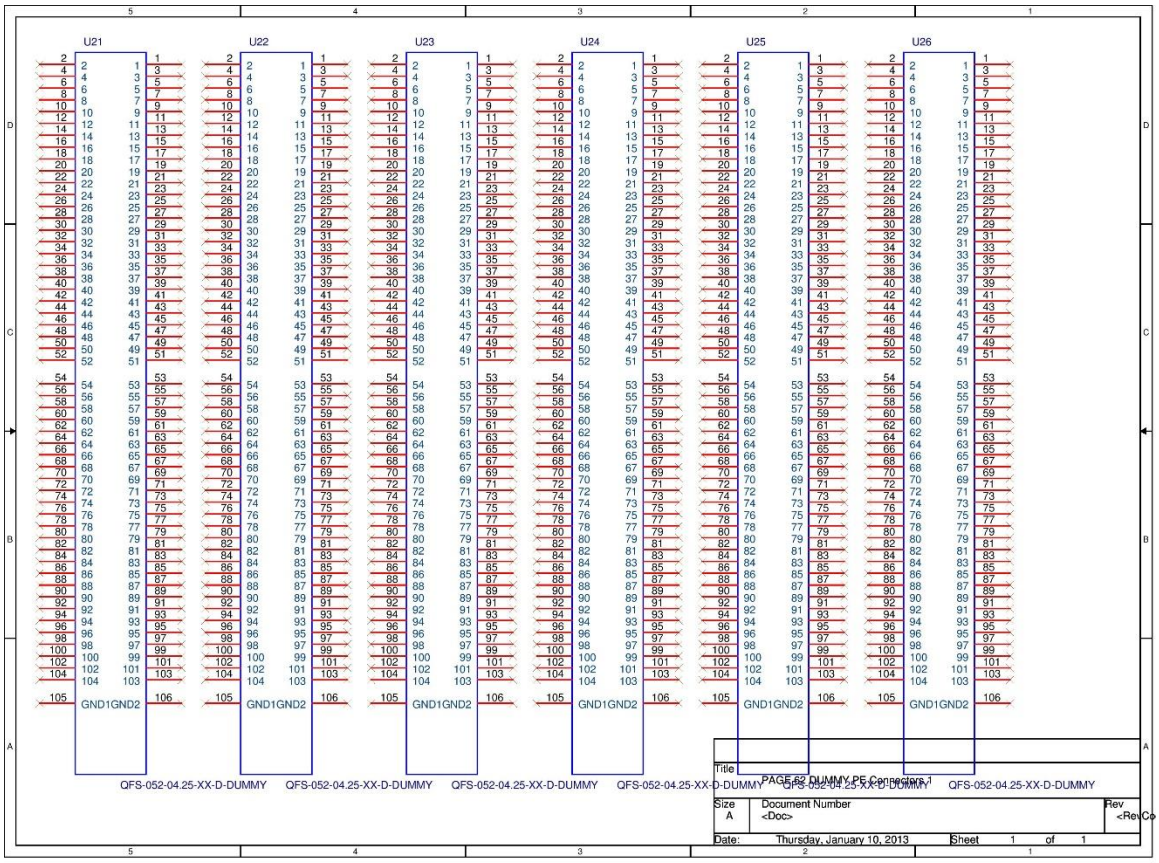

U11
VCC_3V3
VCC_1V5_n
VCC_12V
VCC_5V_n
PE_CONN_C_LS2
PE_CONN_C_LS1
PE_CONN_C_LS4
PE_CONN_C_LS3
PE_CONN_C_SS0_P
PE_CONN_C_SS1_P
PE_CONN_C_SS0_N
PE_CONN_C_SS1_N
PE_CONN_C_SS2_P
PE_CONN_C_SS3_P
PE_CONN_C_SS2_N
PE_CONN_C_SS3_N
PE_CONN_C_SS_CLK_P
PE_CONN_C_SS_CLK_N
Shadow Sampler Outputs and Clk (SS0, SS1, SS2, SS3)
PE_CONN_C_IO22
PE_CONN_C_IO24
PE_CONN_C_IO26
PE_CONN_C_IO28
PE_CONN_C_IO30
PE_CONN_C_IO32
PE_CONN_C_IO34
PE_CONN_C_IO36
PE_CONN_C_IO38
PE_CONN_C_IO40
PE_CONN_C_IO42
PE_CONN_C_IO44
PE_CONN_C_IO46
PE_CONN_C_IO48
PE_CONN_C_IO50
PE_CONN_C_IO52
PE_CONN_C_IO25
PE_CONN_C_IO27
PE_CONN_C_IO29
PE_CONN_C_IO31
PE_CONN_C_IO33
PE_CONN_C_IO35
PE_CONN_C_IO37
PE_CONN_C_IO39
PE_CONN_C_IO41
PE_CONN_C_IO43
PE_CONN_C_IO45
PE_CONN_C_IO47
PE_CONN_C_IO49
PE_CONN_C_IO51
FPGA User I/Os - Bank 18
PE_CONN_C_IO54
PE_CONN_C_IO56
PE_CONN_C_IO58
PE_CONN_C_IO60
PE_CONN_C_IO53
PE_CONN_C_IO55
PE_CONN_C_IO57
PE_CONN_C_IO59
FPGA Gigabit Transceivers
MGT_C0_RX_P
MGT_C0_RX_N
MGT_C3_RX_P
MGT_C3_RX_N
MGT_C0_TX_P
MGT_C0_TX_N
MGT_C3_TX_P
MGT_C3_TX_N
MGT_C1_RX_P
MGT_C1_RX_N
MGT_C2_RX_P
MGT_C2_RX_N
MGT_C1_TX_P
MGT_C1_TX_N
MGT_C2_TX_P
MGT_C2_TX_N
VCC_1V8
VCC_3V_n
VCC_5V
VTERM
VREF_C
GND2GND1
VCC2 VCC1
VCC4 VCC3
VCC6 VCC5
VCC8 VCC7
QFS-052-04.25-L-D-PC4
Title
PAGE 50 PE Connector C
Size
A
Document Number
<Doc>
Rev
<RevCo
Date:
Thursday, January 10, 2013
Sheet
1
of
1

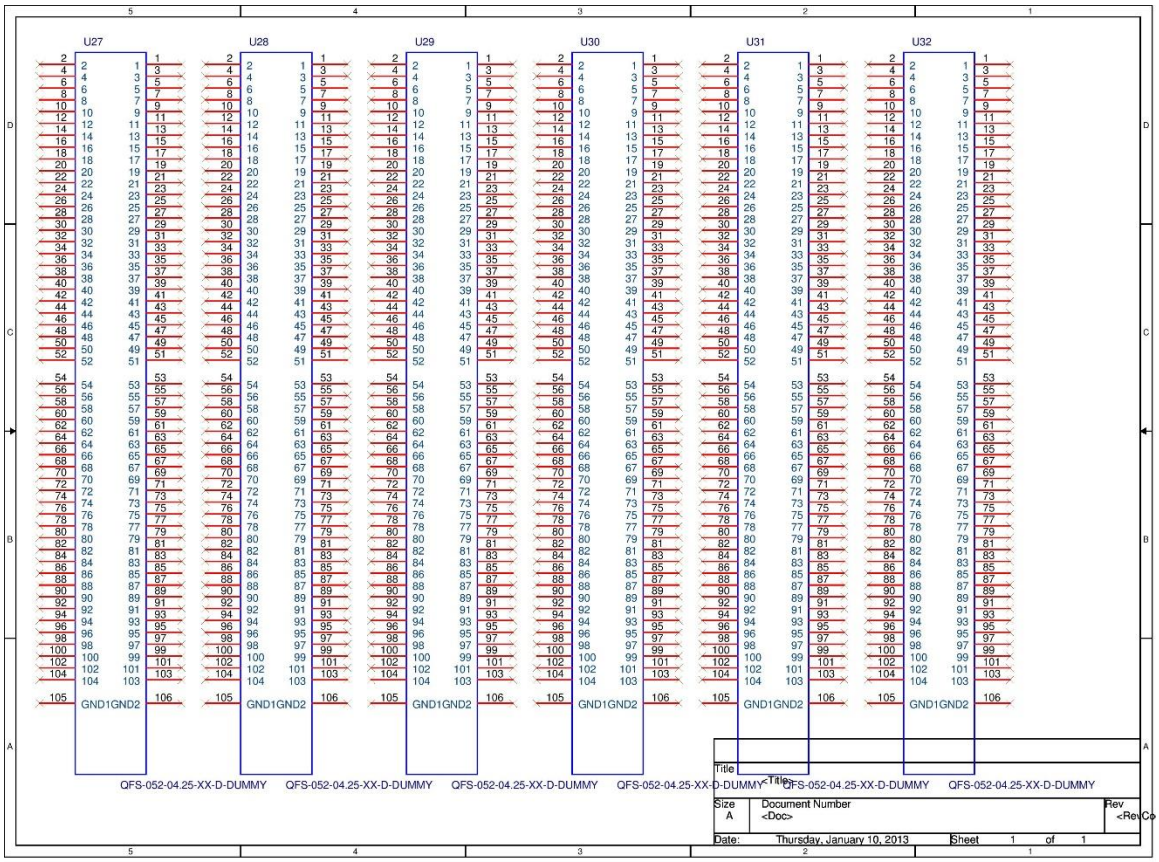

VCC_3V3
U12
VCC_1V5_n
PE_CONN_D_LS2
VCC_12V
VCC_5V_n
PE_CONN_D_LS1
PE_CONN_D_LS4
PE_CONN_D_SS0_P
PE_CONN_D_SS0_N
PE_CONN_D_SS2_P
PE_CONN_D_SS2_N
PE_CONN_D_LS3
PE_CONN_D_SS1_P
PE_CONN_D_SS1_N
PE_CONN_D_SS3_P
PE_CONN_D_SS3_N
Shadow Sampler Outputs and Clk (SS0, SS1, SS2, SS3)
PE_CONN_D_SS_CLK_P
PE_CONN_D_SS_CLK_N
FPGA User I/Os - Bank 17
FPGA Gigabit Transceivers
MGT_D0_RX_P
MGT_D0_RX_N
MGT_D3_RX_P
MGT_D3_RX_N
MGT_D0_TX_P
MGT_D0_TX_N
MGT_D3_TX_P
MGT_D3_TX_N
MGT_D1_RX_P
MGT_D1_RX_N
MGT_D2_RX_P
MGT_D2_RX_N
MGT_D1_TX_P
MGT_D1_TX_N
MGT_D2_TX_P
MGT_D2_TX_N
VCC_1V8
VCC_3V_n
VCC_5V
VTERM
VREF_D
GND2GND1
VCC2 VCC1
VCC4 VCC3
VCC6 VCC5
VCC8 VCC7
QFS-052-04.25-L-D-PC4
Title
PAGE 51 PE Connector D
Size A
Document Number <Doc>
Rev <Rev Co
Date: Thursday, January 10, 2013
Sheet 1 of 1


VCC_12V
VCC_5V
VCC_3V3
VCC_1V8
VCC_1V5_n
VCC_3V_n
VTERM
VCC_1V_FPGA
VCC_1V2_FPGA
VCC_1V8_FPGA
VVHDM_FPGA
R247 0/0402
PPS_S0
PPS_S1
PPS_S2
PPS_S3
PPS_S4
PPS_S5
PPS_S6
PPS_S7
PPS_S8
PPS_S9
PPS_S10
PPS_S11
PPS_S12
PPS_S13
PPS_S14
PPS_S15
PPS_S16
PPS_S17
PPS_S18
PPS_S19
Ferrite/0805
U13
Header40_2.54mm
C72
C73
100uF/1206
C44
C45
100uF/1206
C74
C75
100uF/1206
330uF/TANT
VTERM
C60
C61
100uF/1206
VCC_1V8
C52
C53
100uF/1206
Routing Guideline: Force (PPS_F*R) 1.2 mm / Sense (PPS_S*) 0.3 mm / Spacing 0.2 mm
Power Force/Sense Connection - Byun (01/02/2013)
Title
PAGE 52 Aux Power Conn A
Size A
Document Number <Doc>
Rev <Rev Co
Date: Thursday, January 10, 2013
Sheet 1 of 1


VCC_2V5_FPGA
VCC_NULL1
VCC_NULL2
PE_CONN_A_LS4 PE_CONN_B_LS4 PE_CONN_C_LS4 PE_CONN_D_LS4
PE_CONN_A_LS3 PE_CONN_B_LS3 PE_CONN_C_LS3 PE_CONN_D_LS3
PE_CONN_A_LS2 PE_CONN_B_LS2 PE_CONN_C_LS2 PE_CONN_D_LS2
PE_CONN_A_LS1 PE_CONN_B_LS1 PE_CONN_C_LS1 PE_CONN_D_LS1
R227 0/0402 PPS_S20
R228 0/0402 PPS_S21
R229 0/0402 PPS_S22
R230 0/0402 PPS_S23
PPS_F_R24 R211 0/0402 PPS_S24
PPS_F_R25 R212 0/0402 PPS_S25
PPS_F_R26 R213 0/0402 PPS_S26
PPS_F_R27 R214 0/0402 PPS_S27
PPS_F_R28 R215 0/0402 PPS_S28
PPS_F_R29 R216 0/0402 PPS_S29
PPS_F_R30 R217 0/0402 PPS_S30
PPS_F_R31 R218 0/0402 PPS_S31
PPS_F_R32 R219 0/0402 PPS_S32
PPS_F_R33 R220 0/0402 PPS_S33
PPS_F_R34 R221 0/0402 PPS_S34
PPS_F_R35 R222 0/0402 PPS_S35
PPS_F_R36 R223 0/0402 PPS_S36
PPS_F_R37 R224 0/0402 PPS_S37
PPS_F_R38 R225 0/0402 PPS_S38
PPS_F_R39 R226 0/0402 PPS_S39
Ferrite/0805
U14
Header40_2.54MM
PPS_F_R[39:24]
VCC_1V5_n
C204 330uF/TANT
C207 100uF/1206
VCC_3V_n
C46 330uF/TANT
C47 100uF/1206
VCC_5V_n
C205 330uF/TANT
C208 100uF/1206
VVHDM_FPGA
C62 330uF/TANT
C206 100uF/1206
VCC_1V_FPGA
C54 330uF/TANT
C55 100uF/1206
VCC_1V2_FPGA
C56 330uF/TANT
C59 100uF/1206
VCC_2V5_FPGA
C214 330uF/TANT
C215 100uF/1206
VCC_1V8_FPGA
C212 330uF/TANT
C213 100uF/1206
Power Force/Sense Connection - Ryun (01/02/2013)
Routing Guideline: Force (PPS_F*R) 1.2 mm / Sense (PPS_S*) 0.3 mm / Spacing 0.2 mm
Title <Title>
Size A
Document Number <Doc>
Rev <RevCo
Date: Thursday, January 10, 2013
Sheet 1 of 1


VCC_1V2_FPGA
VCC_2V5_FPGA
VCC_1V8_FPGA
VCC_1V2_FPGA
VCC_1V_FPGA
VCC_1V_FPGA
VVHDM_FPGA
VCC_1V8_FPGA
VCC_1V_FPGA
VCC_1V8_FPGA
VCC_2V5_FPGA
VCC_2V5_FPGA
VCC_2V5_FPGA
VCC_2V5_FPGA
VCC_1V8_FPGA
Re-Assigned capacitors
Title PAGE 53 Aux Power Conn D
Size Custom
Document Number <Doc>
Rev <RevCo
Date: Friday, January 18, 2013
Sheet 1 of 1


U21
U22
U23
U24
U25
U26
GND1GND2
QFS-052-04.25-XX-D-DUMMY
PAGE 62 DUMMY PE Connectors 1
Title
Size A
Document Number <Doc>
Rev <Rev
Date: Thursday, January 10, 2013
Sheet 1 of 1


U27
U28
U29
U30
U31
U32
GND1GND2
QFS-052-04.25-XX-D-DUMMY
Title
Size A
Document Number <Doc>
Rev <Rev
Date: Thursday, January 10, 2013
Sheet 1 of 1

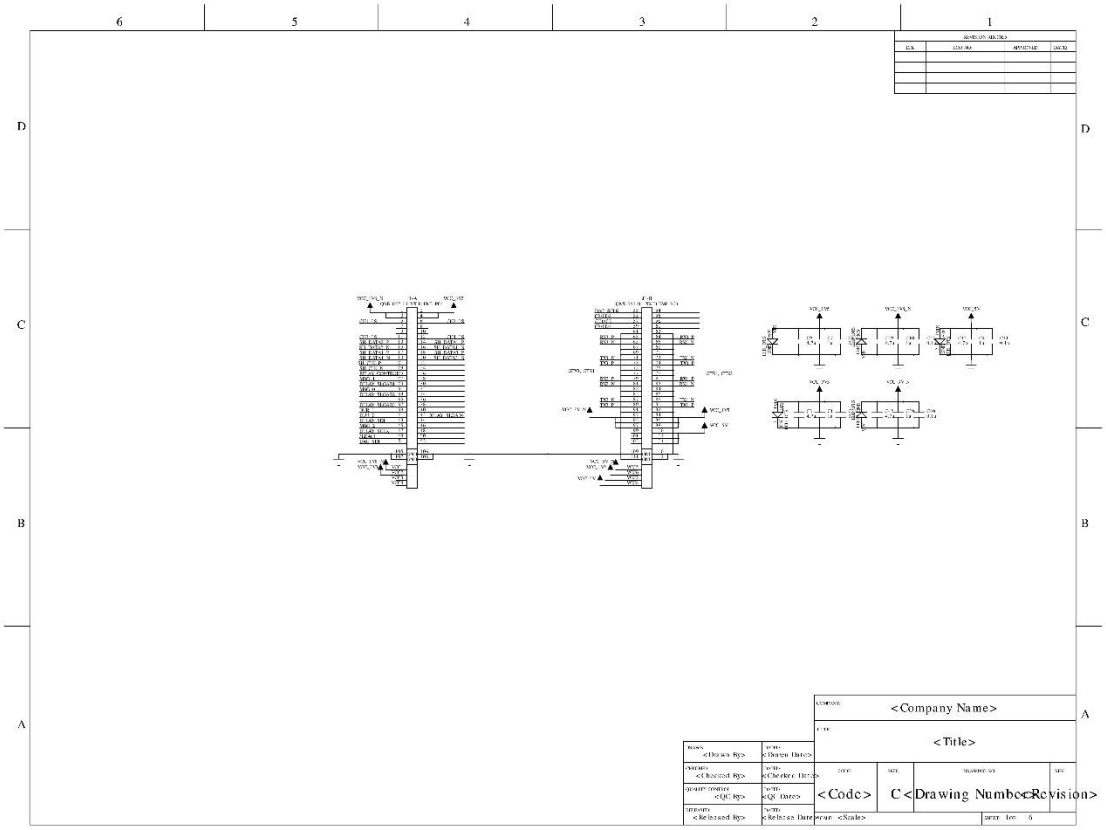
APPENDIX B

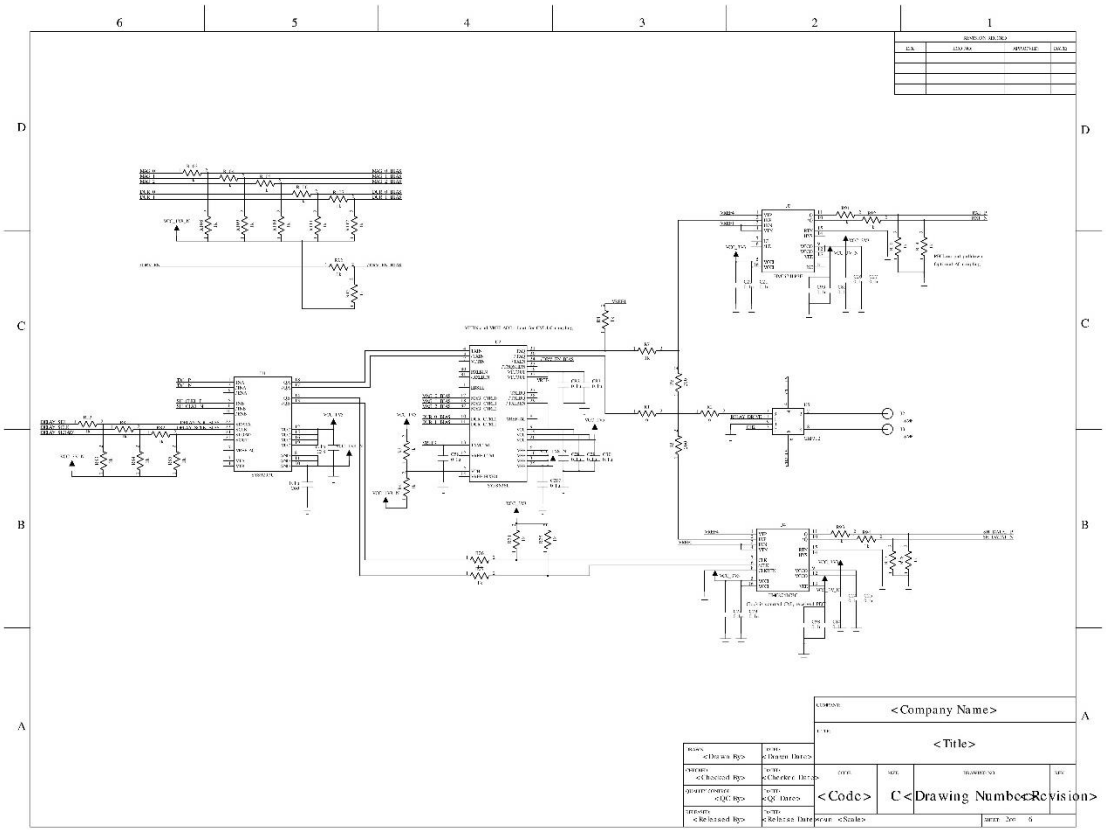
EXTENSION BAORD DESIGN AND LAYOUT

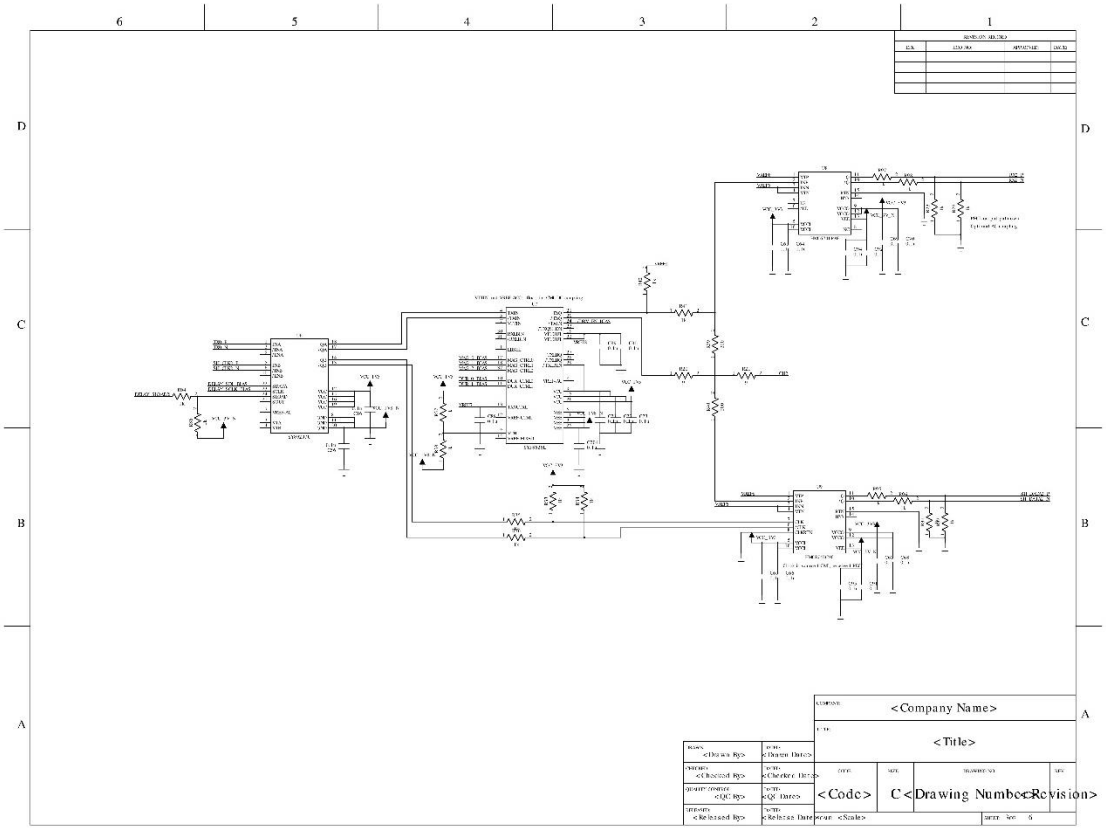
This appendix illustrates the schematic design and layout of the two extension board in this thesis. In the first part the schematic and the layout of PE board is shown and follows the design of UHS testing board.

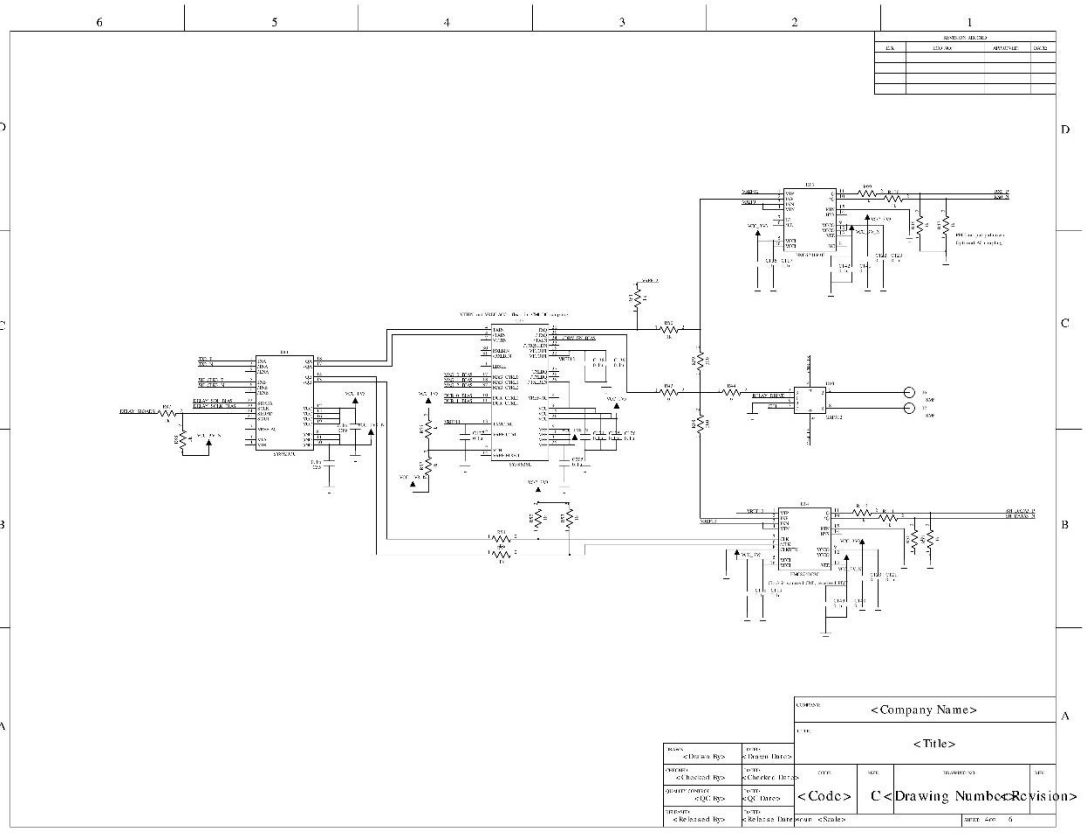
B.1 PE Board

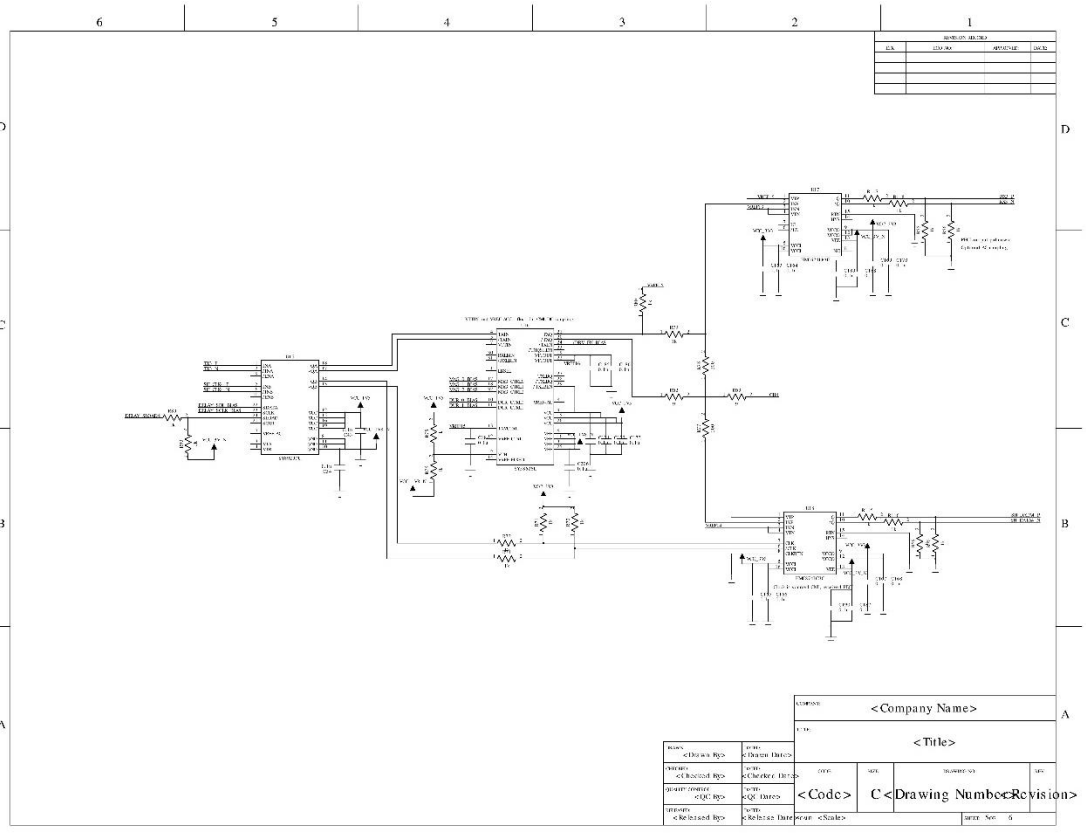
Schematics

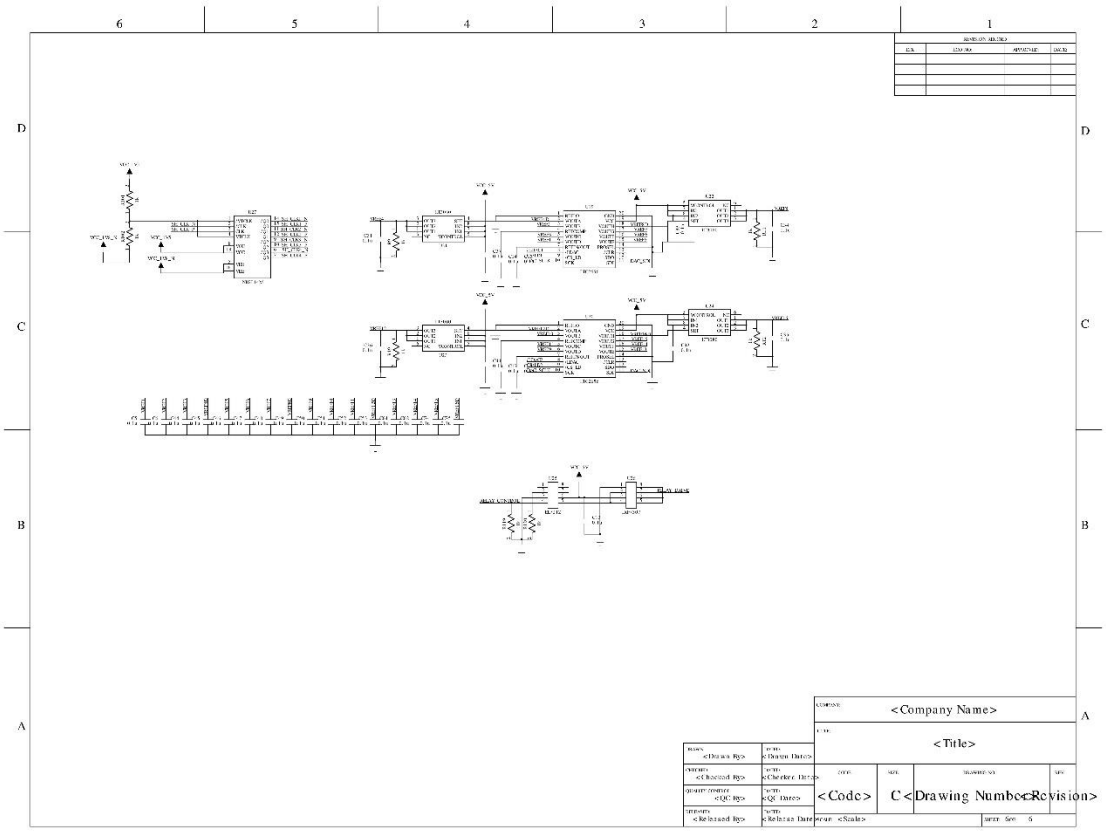



<Company Name>
<Title>
<Code>
C
<Drawing Number>
<Revision>

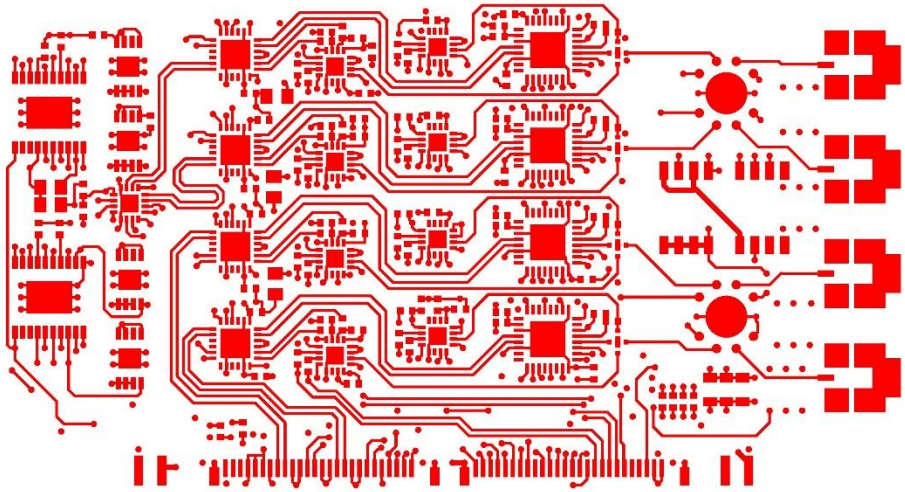

<Company Name>
<Title>
<Code>
C
<Drawing Number>
<Revision>


<Company Name>
<Title>
<Code>
<Drawing Number>
<Revision>

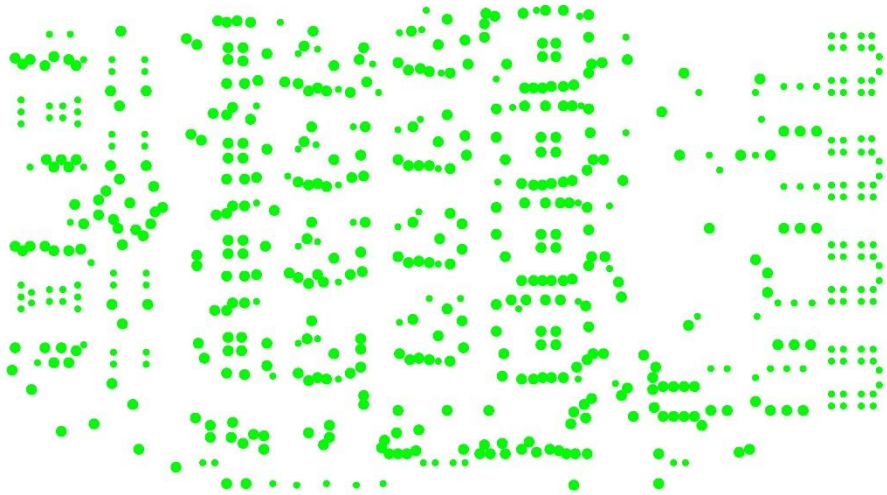

<Company Name>
<Title>
<Code>
<Drawing Number>
<Revision>


<Company Name>
<Title>
<Code>
C
<Drawing Number>
<Revision>

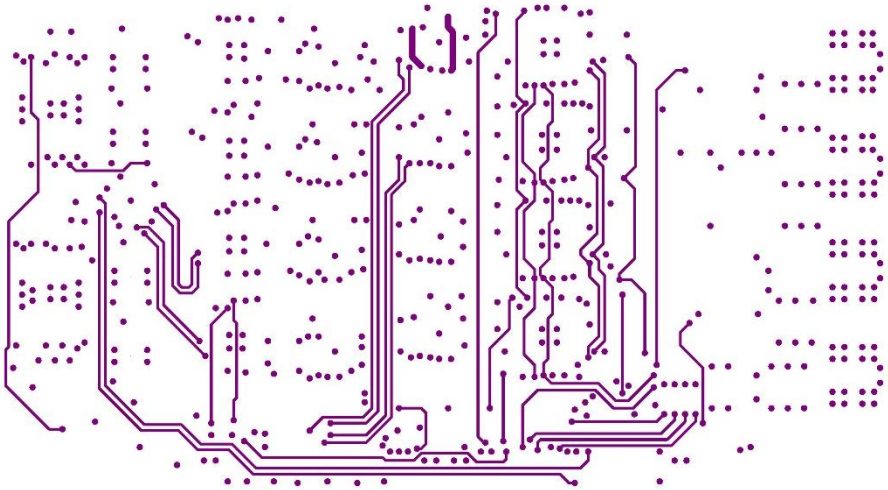
Layout



Layer 1: Top layer



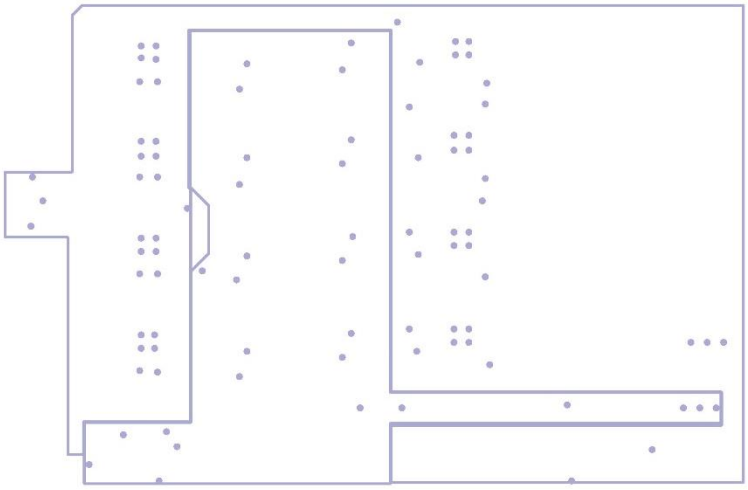
Layer 2: Ground 1



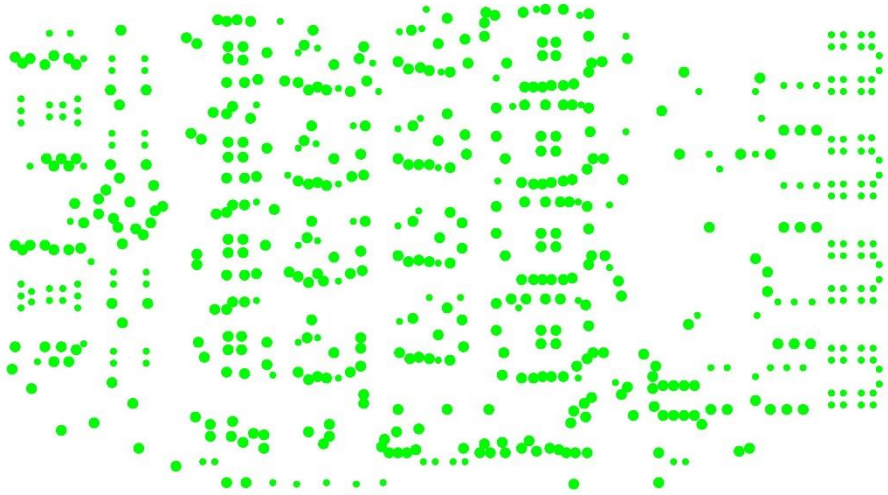
Layer 3: Inner 1

Layer 4: Ground 2

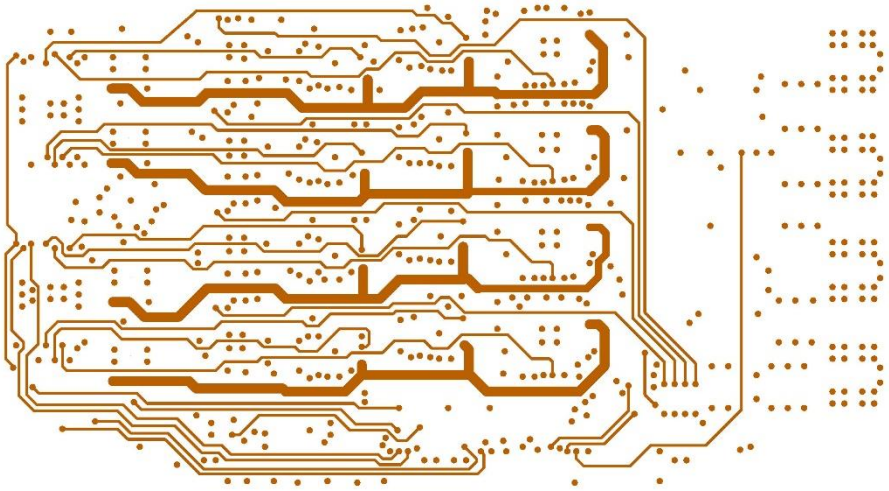
Layer 5: Power 1



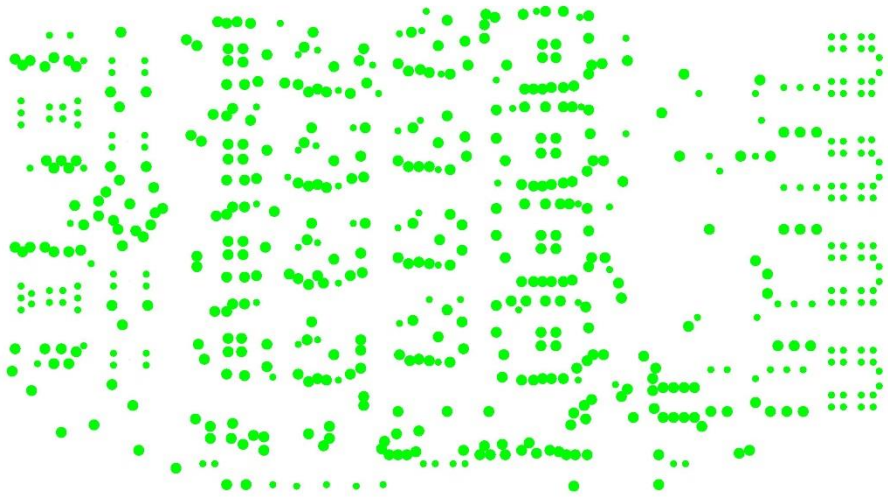
Layer 6: Power 2



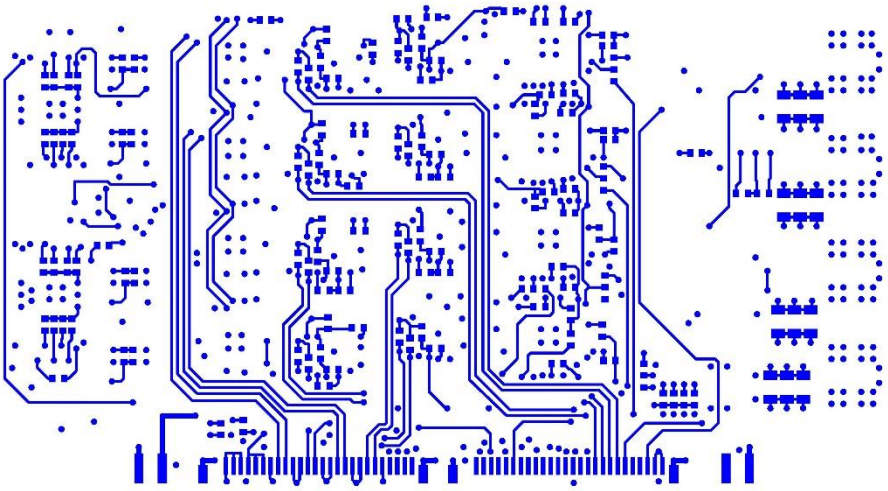
Layer 7: Ground 3



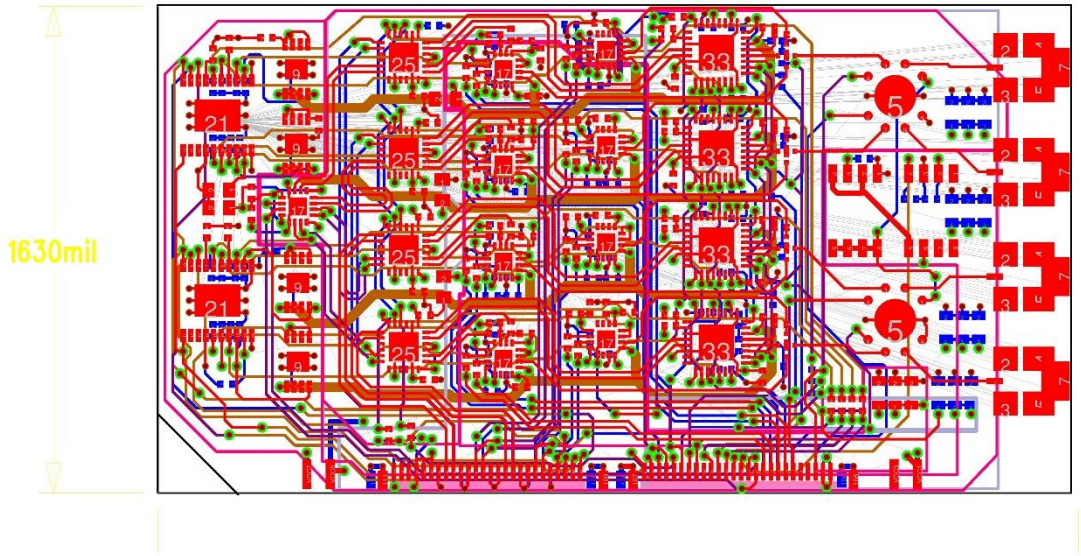
Layer 8: Inner 2



Layer 9: Ground 4



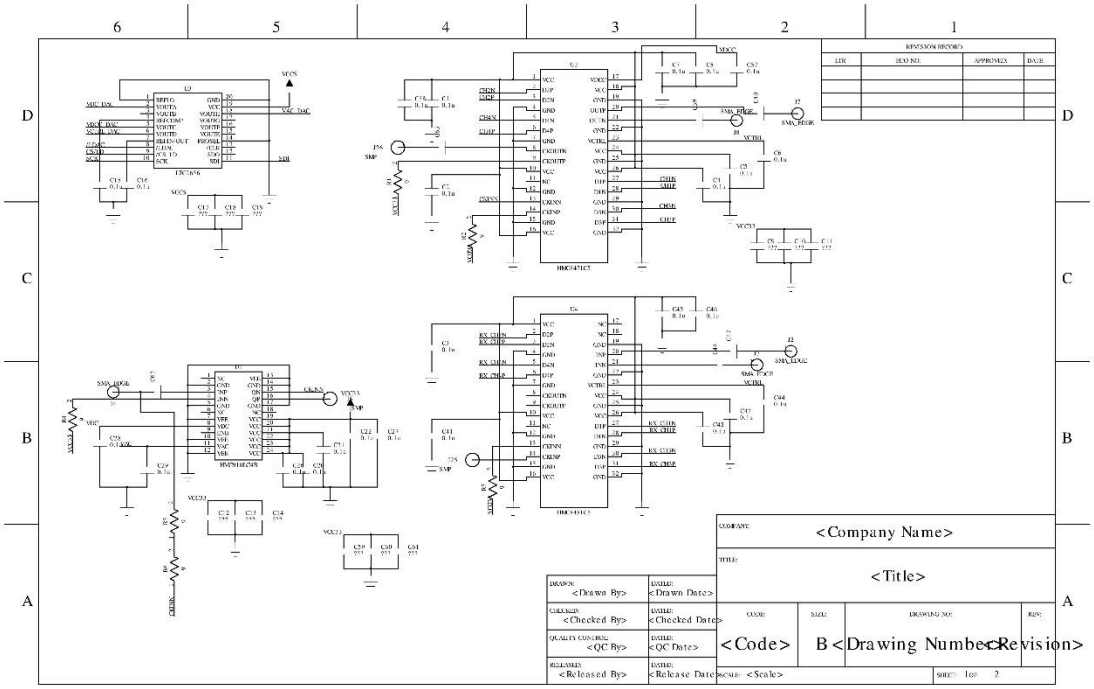
Layer 10: Bottom layer

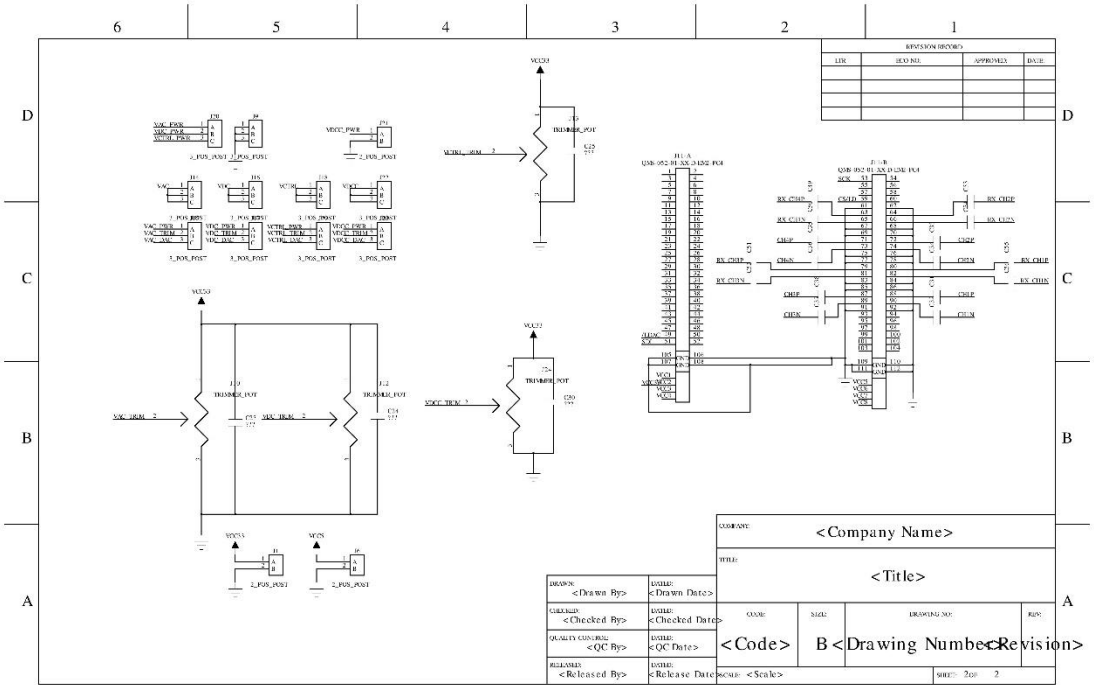


Composite layer

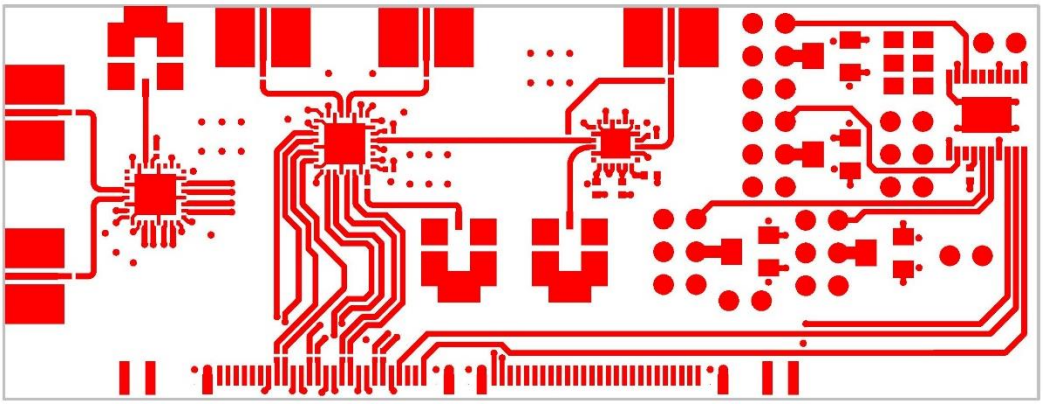
B.2 Ultra-High-Speed testing Board

Schematics

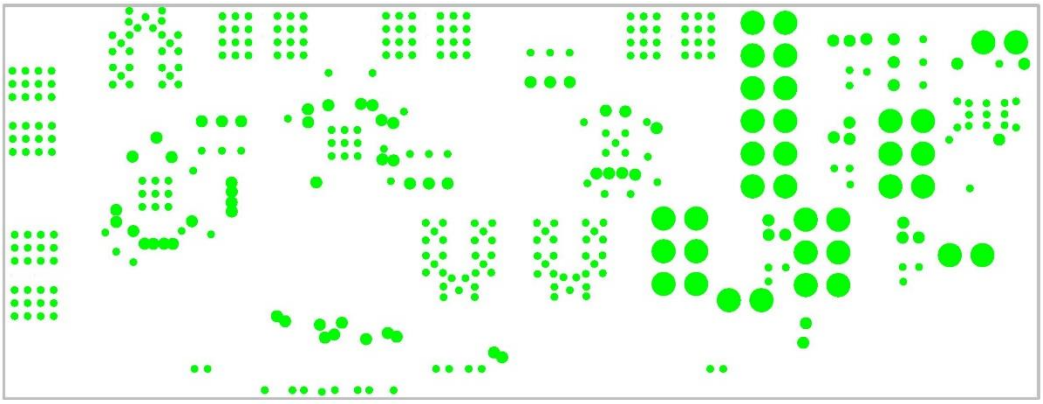




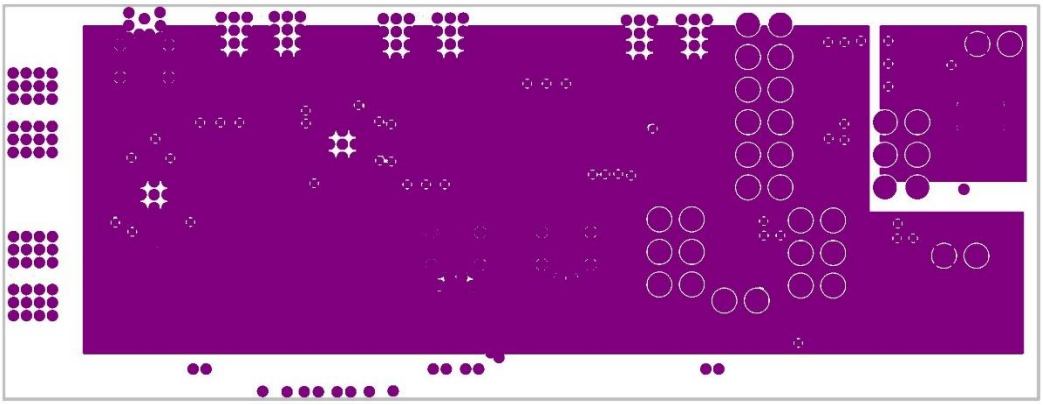
Layout



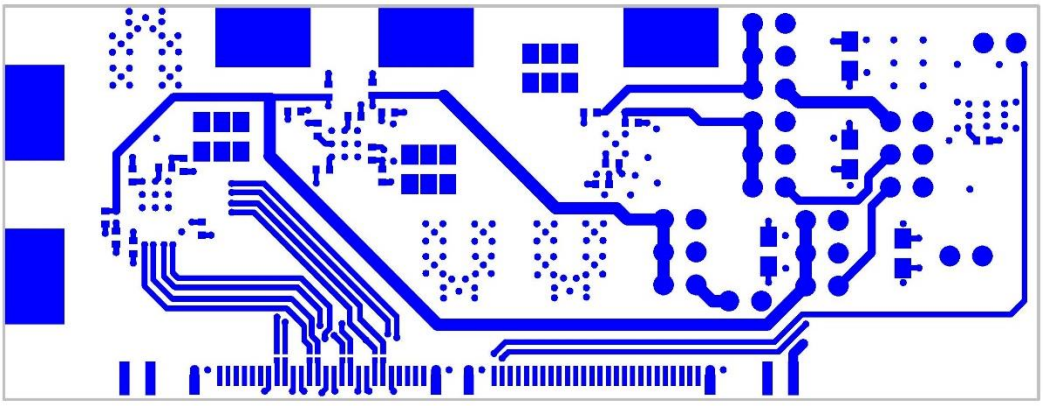
Layer 1: Top layer



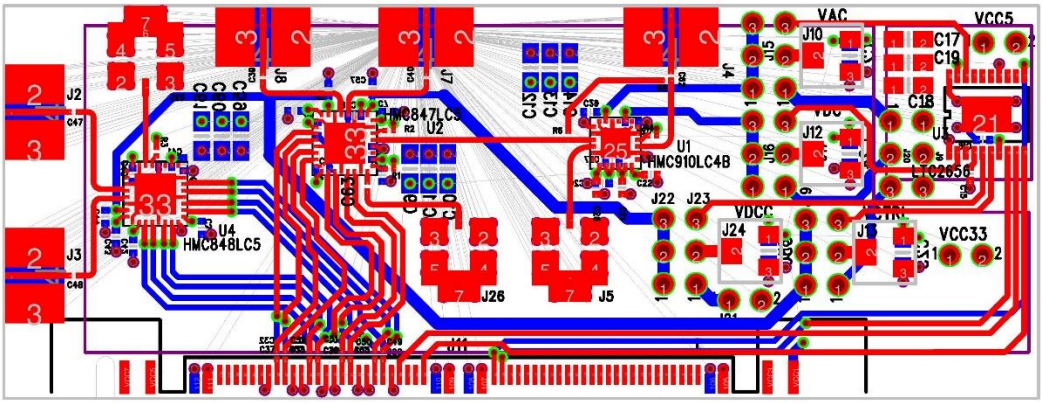
Layer 2: Ground



Layer 3: Power



Layer 4: Bottom layer



Composite layer

APPENDIX C

FPGA-BASED TESTING PLATFORM FIRMWARE

In this appendix detailed the FPGA-based testing platform firmware implemented by Verilog language. The code is compiled by Xilinx ISE13.0 software and programmed into FPGA, and the new function can always to be added.

```
module socTop (
    sys_clk_p,
    sys_clk_n,

    usr_clk_p,
    usr_clk_n,

    pb_rst,

    rs232tx,
    rs232rx,

    gpo,

    switch,
    led,

    txp,
    txn,

    rxp,
    rxn
);

parameter EXAMPLE_SIMULATION = 0;

input sys_clk_p;
input sys_clk_n;

input usr_clk_p;
input usr_clk_n;

input pb_rst;

output rs232tx;
input  rs232rx;

output [3:0] gpo;

input  [4:0] switch;
output [4:0] led;

output [7:0] txp;
output [7:0] txn;

input  [7:0] rxp;
input  [7:0] rxn;

wire pb_rst_n = ~pb_rst;

reg [3:0] sysClkCnt;
```

```
wire sysClkDiv4;

wire [31:0] unused_txp;
wire [31:0] unused_txn;

wire rs232rx_sync;
wire [4:0] switch_sync;

reg   [7:0] pb_debounce_count;
reg         early_pb_rst_n;
reg         user_pb_rst_n;
reg   [7:0] user_wait_count;

reg  [25:0] heartbeatCnt;
reg         sysRstN;
reg         softRstN;
reg         asRstN;
reg         txRstN;
reg         rxRstN;


//each led is active when driven with 1'b1
assign      led[0] = ~sysRstN;
assign      led[1] = ~softRstN;
assign      led[2] = sysRstN & heartbeatCnt[25];
assign      led[3] = sysRstN & rs232rx_sync;
assign      led[4] = sysRstN & rs232tx;

wire sys_rst_disable  = switch_sync[0];
wire soft_rst_disable = switch_sync[1];
wire as_rst_disable   = switch_sync[2];
wire tx_rst_disable   = switch_sync[3];
wire rx_rst_disable   = switch_sync[4];

//calculation: clk freq / baud rate --> 50000000/115200 = 434.03; round to nearest integer (434).
wire [15:0] uart_divisor = 16'd434;

socSystem #(.EXAMPLE_SIMULATION(EXAMPLE_SIMULATION)) socSystem (

   .usr_clk_p    (usr_clk_p),
   .usr_clk_n    (usr_clk_n),

   .sysRstN      (sysRstN),
   .softRstN     (softRstN),
   .asRstN       (asRstN),
   .txRstN       (txRstN),
   .rxRstN       (rxRstN),

   .sysClk       (sysClkDiv4),      //50MHz
   .uart_divisor (uart_divisor),
   .rs232rx      (rs232rx_sync),
   .rs232tx      (rs232tx),

   .gpo          (gpo),

   .txp          ({unused_txp[31:8], txp}),
   .txn          ({unused_txn[31:8], txn}),

   .rxp          ({24'd0, rxp}),
   .rxn          ({24'd0, rxn})
);

/*IBUFGDS ibufgds_sysClk
(
   .O             (sysClk),
   .I             (sys_clk_p),
   .IB            (sys_clk_n)
);*/

IBUFG sysclk
```

```
(
   .O (sysClk),
   .I (sys_clk_p)
);



//early reset synchronization to system clock
socSync #(.SIGNAL_WIDTH(1), .INIT_VAL(0)) i_socSync_early_pb_rst_n_sync (
   .outClk     (sysClkDiv4),
   .outRstN    (early_pb_rst_n),
   .asyncInput (early_pb_rst_n),
   .syncOutput (earlyRstN)
);

//user reset synchronization to system clock
socSync #(.SIGNAL_WIDTH(1), .INIT_VAL(0)) i_socSync_user_pb_rst_n_sync (
   .outClk     (sysClkDiv4),
   .outRstN    (early_pb_rst_n),
   .asyncInput (user_pb_rst_n),
   .syncOutput (userRstN)
);

//switch reset synchronization
socSync #(.SIGNAL_WIDTH(5), .INIT_VAL(5'b11111)) i_socSync_switch_sync (
   .outClk     (sysClkDiv4),
   .outRstN    (earlyRstN),
   .asyncInput (switch),
   .syncOutput (switch_sync)
);

//uart receive synchronization
socSync #(.SIGNAL_WIDTH(1), .INIT_VAL(1)) i_socSync_rs232rx_sync (
   .outClk     (sysClkDiv4),
   .outRstN    (sysRstN),
   .asyncInput (rs232rx),
   .syncOutput (rs232rx_sync)
);

//reset debuounce logic, generation of early, user versions of reset button push
always @ (posedge sysClk or negedge pb_rst_n) begin
   if (!pb_rst_n) begin
      pb_debounce_count  <= 8'd0;
      early_pb_rst_n     <= 1'b0;
      user_pb_rst_n      <= 1'b0;
      user_wait_count    <= 8'd0;
   end
   else begin
      if (~(&pb_debounce_count)) begin
         pb_debounce_count <= pb_debounce_count + 1;
         early_pb_rst_n    <= 1'b0;
         user_pb_rst_n     <= 1'b0;
         user_wait_count   <= 8'd0;
      end
      else begin
         //pb_debounce_count has reached its limit - stop counting and release the early reset, which begins the toggling of the system
clock
         early_pb_rst_n <= 1'b1;

         //start the user_wait_count (giving the system clock some cycles to toggle) and release user reset when the count reaches its limit
         if (~(&user_wait_count)) begin
            user_wait_count   <= user_wait_count + 1;
            user_pb_rst_n <= 1'b0;
         end
         else begin
            user_pb_rst_n <= 1'b1;
         end
      end
   end
end
```

```
//sysClkDiv4 generation
always @ (posedge sysClk or negedge early_pb_rst_n) begin
   if (!early_pb_rst_n) begin
      sysClkCnt <= 4'b0000;
   end
   else begin
      sysClkCnt <= sysClkCnt + 1;
   end
end
BUFG BUFG_clk4 (
     .O(sysClkDiv4), // 1-bit output: Clock output
     .I(sysClkCnt[1])  // 1-bit input: Clock input
  );
//heartbeat generation logic, reset signal generation
always @ (posedge sysClkDiv4 or negedge earlyRstN) begin
   if (~earlyRstN) begin
      heartbeatCnt <= 26'd0;

      sysRstN     <= 1'b1;
      softRstN    <= 1'b1;
      asRstN      <= 1'b1;
      txRstN      <= 1'b1;
      rxRstN      <= 1'b1;
   end
endmodule


module socSystem (
   sysRstN,
   softRstN,
   asRstN,
   txRstN,
   rxRstN,

   sysClk,

   uart_divisor,
   rs232rx,
   rs232tx,

   usr_clk_p,
   usr_clk_n,

   gpo,

   txp,
   txn,

   rxp,
   rxn
);

parameter EXAMPLE_SIMULATION = 0;

input  sysRstN;
input  softRstN;
input  asRstN;
input  txRstN;
input  rxRstN;

input  sysClk;

input  [15:0] uart_divisor;
input  rs232rx;
output rs232tx;

input  usr_clk_p;
input  usr_clk_n;
```

```
output  [3:0] gpo;

output [31:0] txp;
output [31:0] txn;

input  [31:0] rxp;
input  [31:0] rxn;

////////////////////////
// DRP Channel Master
////////////////////////

wire [31:0] DRPADDRM;
wire        DRPSELM;
wire        DRPWRITEM;
wire [63:0] DRPWDATAM;
wire [63:0] DRPRDATAM;
wire        DRPREADYM;

////////////////////////
// DRP Channel Port 0
////////////////////////

wire        DRPCLK0;
wire        DRPRESET0n;
wire [31:0] DRPADDR0;
wire        DRPSEL0;
wire        DRPWRITE0;
wire [63:0] DRPWDATA0;
wire [63:0] DRPRDATA0;
wire        DRPREADY0;

wire        X_DRPCLK0;
wire        X_DRPRESET0n;
wire [31:0] X_DRPADDR0;
wire        X_DRPSEL0;
wire        X_DRPWRITE0;
wire [63:0] X_DRPWDATA0;
wire [63:0] X_DRPRDATA0;
wire        X_DRPREADY0;

wire [31:0] asBroadcast0;
wire  [3:0] asGpoMuxSel0;
wire        asSoftRst0;
wire  [2:0] asLoopback0;
wire  [1:0] asTxSysClkSel0;
wire  [1:0] asRxSysClkSel0;
wire  [2:0] asTxOutClkSel0;
wire  [2:0] asRxOutClkSel0;
wire  [3:0] asTxDiffCtrl0;
wire  [6:0] asTxMainCursor0;
wire  [4:0] asTxPreCursor0;
wire  [4:0] asTxPostCursor0;

wire [63:0] txData0;
wire  [2:0] txPrbsSel0;
wire  [1:0] txpdin0;
wire        txPrbsForceErrLvl0;
wire        txPrbsForceErrPls0;
wire        txPolarity0;
wire        txResetDone0;
wire        tx8b10bEn0;
wire  [2:0] txRate0;

wire [63:0] rxData0;
wire  [2:0] rxPrbsSel0;
wire        rxPrbsCntRstPls0;
wire        rxPolarity0;
wire        rxPrbsErrFlag0;
wire        rxResetDone0;
```

```
wire       rx8b10bEn0;
wire  [2:0] rxRate0;

//////////////////////////
// DRP Channel Port 1
//////////////////////////

wire       DRPCLK1;
wire       DRPRESET1n;
wire [31:0] DRPADDR1;
wire       DRPSEL1;
wire       DRPWRITE1;
wire [63:0] DRPWDATA1;
wire [63:0] DRPRDATA1;
wire       DRPREADY1;

wire       X_DRPCLK1;
wire       X_DRPRESET1n;
wire [31:0] X_DRPADDR1;
wire       X_DRPSEL1;
wire       X_DRPWRITE1;
wire [63:0] X_DRPWDATA1;
wire [63:0] X_DRPRDATA1;
wire       X_DRPREADY1;

wire  [2:0] asLoopback1;
wire  [1:0] asTxSysClkSel1;
wire  [1:0] asRxSysClkSel1;
wire  [2:0] asTxOutClkSel1;
wire  [2:0] asRxOutClkSel1;
wire  [3:0] asTxDiffCtrl1;
wire  [6:0] asTxMainCursor1;
wire  [4:0] asTxPreCursor1;
wire  [4:0] asTxPostCursor1;

wire [63:0] txData1;
wire  [2:0] txPrbsSel1;
wire  [1:0] txpdin1;
wire       txPrbsForceErrLvl1;
wire       txPrbsForceErrPls1;
wire       txPolarity1;
wire       txResetDone1;
wire       tx8b10bEn1;
wire  [2:0] txRate1;

wire [63:0] rxData1;
wire  [2:0] rxPrbsSel1;
wire       rxPrbsCntRstPls1;
wire       rxPolarity1;
wire       rxPrbsErrFlag1;
wire       rxResetDone1;
wire       rx8b10bEn1;
wire  [2:0] rxRate1;

//////////////////////////
// DRP Channel Port 2
//////////////////////////

wire       DRPCLK2;
wire       DRPRESET2n;
wire [31:0] DRPADDR2;
wire       DRPSEL2;
wire       DRPWRITE2;
wire [63:0] DRPWDATA2;
wire [63:0] DRPRDATA2;
wire       DRPREADY2;

wire       X_DRPCLK2;
wire       X_DRPRESET2n;
wire [31:0] X_DRPADDR2;
```

```
wire       X_DRPSEL2;
wire       X_DRPWRITE2;
wire [63:0] X_DRPWDATA2;
wire [63:0] X_DRPRDATA2;
wire       X_DRPREADY2;

wire  [2:0] asLoopback2;
wire  [1:0] asTxSysClkSel2;
wire  [1:0] asRxSysClkSel2;
wire  [2:0] asTxOutClkSel2;
wire  [2:0] asRxOutClkSel2;
wire  [3:0] asTxDiffCtrl2;
wire  [6:0] asTxMainCursor2;
wire  [4:0] asTxPreCursor2;
wire  [4:0] asTxPostCursor2;

wire [63:0] txData2;
wire  [2:0] txPrbsSel2;
wire  [1:0] txpdin2;
wire       txPrbsForceErrLvl2;
wire       txPrbsForceErrPls2;
wire       txPolarity2;
wire       txResetDone2;
wire       tx8b10bEn2;
wire  [2:0] txRate2;

wire [63:0] rxData2;
wire  [2:0] rxPrbsSel2;
wire       rxPrbsCntRstPls2;
wire       rxPolarity2;
wire       rxPrbsErrFlag2;
wire       rxResetDone2;
wire       rx8b10bEn2;
wire  [2:0] rxRate2;

//////////////////////////
// DRP Channel Port 3
//////////////////////////

wire       DRPCLK3;
wire       DRPRESET3n;
wire [31:0] DRPADDR3;
wire       DRPSEL3;
wire       DRPWRITE3;
wire [63:0] DRPWDATA3;
wire [63:0] DRPRDATA3;
wire       DRPREADY3;

wire       X_DRPCLK3;
wire       X_DRPRESET3n;
wire [31:0] X_DRPADDR3;
wire       X_DRPSEL3;
wire       X_DRPWRITE3;
wire [63:0] X_DRPWDATA3;
wire [63:0] X_DRPRDATA3;
wire       X_DRPREADY3;

wire  [2:0] asLoopback3;
wire  [1:0] asTxSysClkSel3;
wire  [1:0] asRxSysClkSel3;
wire  [2:0] asTxOutClkSel3;
wire  [2:0] asRxOutClkSel3;
wire  [3:0] asTxDiffCtrl3;
wire  [6:0] asTxMainCursor3;
wire  [4:0] asTxPreCursor3;
wire  [4:0] asTxPostCursor3;

wire [63:0] txData3;
wire  [2:0] txPrbsSel3;
wire  [1:0] txpdin3;
```

```
wire       txPrbsForceErrLvl3;
wire       txPrbsForceErrPls3;
wire       txPolarity3;
wire       txResetDone3;
wire       tx8b10bEn3;
wire  [2:0] txRate3;

wire [63:0] rxData3;
wire  [2:0] rxPrbsSel3;
wire       rxPrbsCntRstPls3;
wire       rxPolarity3;
wire       rxPrbsErrFlag3;
wire       rxResetDone3;
wire       rx8b10bEn3;
wire  [2:0] rxRate3;

////////////////////////
// DRP Channel Port 16 (Common)
////////////////////////

wire       DRPCLK16;
wire       DRPRESET16n;
wire [31:0] DRPADDR16;
wire       DRPSEL16;
wire       DRPWRITE16;
wire [63:0] DRPWDATA16;
wire [63:0] DRPRDATA16;
wire       DRPREADY16;

wire       X_DRPCLK16;
wire       X_DRPRESET16n;
wire [31:0] X_DRPADDR16;
wire       X_DRPSEL16;
wire       X_DRPWRITE16;
wire [63:0] X_DRPWDATA16;
wire [63:0] X_DRPRDATA16;
wire       X_DRPREADY16;

////////////////////////
// DRP Channel Port 17 (Common)
////////////////////////

wire       DRPCLK17;
wire       DRPRESET17n;
wire [31:0] DRPADDR17;
wire       DRPSEL17;
wire       DRPWRITE17;
wire [63:0] DRPWDATA17;
wire [63:0] DRPRDATA17;
wire       DRPREADY17;

wire       X_DRPCLK17;
wire       X_DRPRESET17n;
wire [31:0] X_DRPADDR17;
wire       X_DRPSEL17;
wire       X_DRPWRITE17;
wire [63:0] X_DRPWDATA17;
wire [63:0] X_DRPRDATA17;
wire       X_DRPREADY17;


//JEH check these wires and assignments...

wire [3:0] txUsrClk;
wire [3:0] txUsrClk2;

wire [3:0] rxUsrClk;
wire [3:0] rxUsrClk2;

wire [3:0] txUsrRstN_SYNC;
```

```
wire [3:0] rxUsrRstN_SYNC;

wire rxCdrLock0;
wire rxCdrLock1;
wire rxCdrLock2;
wire rxCdrLock3;

wire GT0_RXOUTCLK_OUT_BUFG; //necessary for par to complete
wire GT0_RXOUTCLK_OUT;
wire GT1_RXOUTCLK_OUT;
wire GT2_RXOUTCLK_OUT;
wire GT3_RXOUTCLK_OUT;

wire GT0_TXOUTCLK_OUT_BUFG; //necessary for par to complete
wire GT0_TXOUTCLK_OUT;
wire GT1_TXOUTCLK_OUT;
wire GT2_TXOUTCLK_OUT;
wire GT3_TXOUTCLK_OUT;


wire gt0_qplllock_i;
wire gt1_qpllock_i = gt0_qpllock_i;
wire gt2_qpllock_i = gt0_qpllock_i;
wire gt3_qpllock_i = gt0_qpllock_i;

assign X_DRPRDATA0[63:16] = 48'd0;
assign X_DRPRDATA1[63:16] = 48'd0;
assign X_DRPRDATA2[63:16] = 48'd0;
assign X_DRPRDATA3[63:16] = 48'd0;


assign X_DRPRDATA16[63:16] = 48'd0;
assign X_DRPRDATA17[63:16] = 48'd0;

assign rxData0[63:32] = 32'd0;
assign rxData1[63:32] = 32'd0;
assign rxData2[63:32] = 32'd0;
assign rxData3[63:32] = 32'd0;


   reg [3:0] gpo;
/*    always @ (*) begin
      case (asGpoMuxSel0)
      4'h0:    gpo = {GT0_RXOUTCLK_OUT_BUFG, GT0_TXOUTCLK_OUT_BUFG};
      4'h1:    gpo = {GT0_RXOUTCLK_OUT_BUFG, rxCdrLock0};
      4'h2:    gpo = {rxCdrLock0,           GT0_TXOUTCLK_OUT_BUFG};
      4'h3:    gpo = 2'b00;*/
//JEH router was unable to route GT4_TXOUTCLK_OUT, GT4_RXOUTCLK_OUT - comment out to see if this was the cause...
/*
      4'h4:    gpo = {GT4_RXOUTCLK_OUT, GT4_TXOUTCLK_OUT};
      4'h5:    gpo = {GT4_RXOUTCLK_OUT, rxCdrLock4};
      4'h6:    gpo = {rxCdrLock4,       GT4_TXOUTCLK_OUT};
      4'h7:    gpo = 2'b00;
*/
      /*default: gpo = 2'b00;
      endcase
   end*/
            always @(*)
            begin
                  gpo[0] <= rxPrbsErrFlag0;
                  gpo[1] <= rxPrbsErrFlag1;
                  gpo[2] <= rxPrbsErrFlag2;
                  gpo[3] <= rxPrbsErrFlag3;
                  //gpo[1] <= GT3_RXOUTCLK_OUT;
            end

   socPxi socPxi (
      .rstN         (sysRstN),
      .clk          (sysClk),
      .sin          (rs232rx),
```

```
    .sout         (rs232tx),
    .pxiDout      (),
    .pxiDin       (),
    .uart_divisor  (uart_divisor),
    .testvector   (),
    .procXferEn    (DRPSELM),
    .procXferDone  (),
    .procWaitN     (DRPREADYM),
    .procAddr      (DRPADDRM),
    .procDin       (DRPRDATAM),
    .procDout      (DRPWDATAM),
    .procWrRdN     (DRPWRITEM),
    .procDsize     ()
);

socDrpChannel socDrpChannel (
    .DRPCLK        (sysClk),
    .DRPRESETn     (sysRstN),
    .DRP_Broadcast (asBroadcast0), //JEH will come from a register...

    .DRPCLKM       (),
    .DRPRESETMn    (),
    .DRPADDRM      (DRPADDRM),
    .DRPSELM       (DRPSELM),
    .DRPWRITEM     (DRPWRITEM),
    .DRPWDATAM     (DRPWDATAM),
    .DRPRDATAM     (DRPRDATAM),
    .DRPREADYM     (DRPREADYM),

    .DRPCLKS0      (DRPCLK0),
    .DRPRESETS0n   (DRPRESET0n),
    .DRPADDRS0     (DRPADDR0),
    .DRPSELS0      (DRPSEL0),
    .DRPWRITES0    (DRPWRITE0),
    .DRPWDATAS0    (DRPWDATA0),
    .DRPRDATAS0    (DRPRDATA0),
    .DRPREADYS0    (DRPREADY0),

    .DRPCLKS1      (DRPCLK1),
    .DRPRESETS1n   (DRPRESET1n),
    .DRPADDRS1     (DRPADDR1),
    .DRPSELS1      (DRPSEL1),
    .DRPWRITES1    (DRPWRITE1),
    .DRPWDATAS1    (DRPWDATA1),
    .DRPRDATAS1    (DRPRDATA1),
    .DRPREADYS1    (DRPREADY1),

    .DRPCLKS2      (DRPCLK2),
    .DRPRESETS2n   (DRPRESET2n),
    .DRPADDRS2     (DRPADDR2),
    .DRPSELS2      (DRPSEL2),
    .DRPWRITES2    (DRPWRITE2),
    .DRPWDATAS2    (DRPWDATA2),
    .DRPRDATAS2    (DRPRDATA2),
    .DRPREADYS2    (DRPREADY2),

    .DRPCLKS3      (DRPCLK3),
    .DRPRESETS3n   (DRPRESET3n),
    .DRPADDRS3     (DRPADDR3),
    .DRPSELS3      (DRPSEL3),
    .DRPWRITES3    (DRPWRITE3),
    .DRPWDATAS3    (DRPWDATA3),
    .DRPRDATAS3    (DRPRDATA3),
    .DRPREADYS3    (DRPREADY3),

    .DRPCLKS16     (DRPCLK16),
    .DRPRESETS16n  (DRPRESET16n),
    .DRPADDRS16    (DRPADDR16),
    .DRPSELS16     (DRPSEL16),
    .DRPWRITES16   (DRPWRITE16),
```

```
.DRPWDATAS16   (DRPWDATA16),
.DRPRDATAS16   (DRPRDATA16),
.DRPREADYS16   (DRPREADY16),

.DRPCLKS17     (DRPCLK17),
.DRPRESETS17n  (DRPRESET17n),
.DRPADDRS17    (DRPADDR17),
.DRPSELS17     (DRPSEL17),
.DRPWRITES17   (DRPWRITE17),
.DRPWDATAS17   (DRPWDATA17),
.DRPRDATAS17   (DRPRDATA17),
.DRPREADYS17   (DRPREADY17),

.DRPCLKS18     (),
.DRPRESETS18n  (),
.DRPADDRS18    (),
.DRPSELS18     (),
.DRPWRITES18   (),
.DRPWDATAS18   (),
.DRPRDATAS18   (64'h1212121212121212),
.DRPREADYS18   (1'b1       ),

.DRPCLKS19     (),
.DRPRESETS19n  (),
.DRPADDRS19    (),
.DRPSELS19     (),
.DRPWRITES19   (),
.DRPWDATAS19   (),
.DRPRDATAS19   (64'h1313131313131313),
.DRPREADYS19   (1'b1       ),

.DRPCLKS20     (),
.DRPRESETS20n  (),
.DRPADDRS20    (),
.DRPSELS20     (),
.DRPWRITES20   (),
.DRPWDATAS20   (),
.DRPRDATAS20   (64'h1414141414141414),
.DRPREADYS20   (1'b1       ),

.DRPCLKS21     (),
.DRPRESETS21n  (),
.DRPADDRS21    (),
.DRPSELS21     (),
.DRPWRITES21   (),
.DRPWDATAS21   (),
.DRPRDATAS21   (64'h1515151515151515),
.DRPREADYS21   (1'b1       ),

.DRPCLKS22     (),
.DRPRESETS22n  (),
.DRPADDRS22    (),
.DRPSELS22     (),
.DRPWRITES22   (),
.DRPWDATAS22   (),
.DRPRDATAS22   (64'h1616161616161616),
.DRPREADYS22   (1'b1       ),

.DRPCLKS23     (),
.DRPRESETS23n  (),
.DRPADDRS23    (),
.DRPSELS23     (),
.DRPWRITES23   (),
.DRPWDATAS23   (),
.DRPRDATAS23   (64'h1717171717171717),
.DRPREADYS23   (1'b1       ),

.DRPCLKS24     (),
.DRPRESETS24n  (),
.DRPADDRS24    (),
```

```
    .DRPSELS24    (),
    .DRPWRITES24  (),
    .DRPWDATAS24  (),
    .DRPRDATAS24  (64'h1818181818181818),
    .DRPREADYS24  (1'b1       ),

    .DRPCLKS25    (),
    .DRPRESETS25n (),
    .DRPADDRS25   (),
    .DRPSELS25    (),
    .DRPWRITES25  (),
    .DRPWDATAS25  (),
    .DRPRDATAS25  (64'h1919191919191919),
    .DRPREADYS25  (1'b1       ),

    .DRPCLKS26    (),
    .DRPRESETS26n (),
    .DRPADDRS26   (),
    .DRPSELS26    (),
    .DRPWRITES26  (),
    .DRPWDATAS26  (),
    .DRPRDATAS26  (64'h1A1A1A1A1A1A1A1A),
    .DRPREADYS26  (1'b1       ),

    .DRPCLKS27    (),
    .DRPRESETS27n (),
    .DRPADDRS27   (),
    .DRPSELS27    (),
    .DRPWRITES27  (),
    .DRPWDATAS27  (),
    .DRPRDATAS27  (64'h1B1B1B1B1B1B1B1B),
    .DRPREADYS27  (1'b1       ),

    .DRPCLKS28    (),
    .DRPRESETS28n (),
    .DRPADDRS28   (),
    .DRPSELS28    (),
    .DRPWRITES28  (),
    .DRPWDATAS28  (),
    .DRPRDATAS28  (64'h1C1C1C1C1C1C1C1C),
    .DRPREADYS28  (1'b1       ),

    .DRPCLKS29    (),
    .DRPRESETS29n (),
    .DRPADDRS29   (),
    .DRPSELS29    (),
    .DRPWRITES29  (),
    .DRPWDATAS29  (),
    .DRPRDATAS29  (64'h1D1D1D1D1D1D1D1D),
    .DRPREADYS29  (1'b1       ),

    .DRPCLKS30    (),
    .DRPRESETS30n (),
    .DRPADDRS30   (),
    .DRPSELS30    (),
    .DRPWRITES30  (),
    .DRPWDATAS30  (),
    .DRPRDATAS30  (64'h1E1E1E1E1E1E1E1E),
    .DRPREADYS30  (1'b1       ),

    .DRPCLKS31    (),
    .DRPRESETS31n (),
    .DRPADDRS31   (),
    .DRPSELS31    (),
    .DRPWRITES31  (),
    .DRPWDATAS31  (),
    .DRPRDATAS31  (64'h1F1F1F1F1F1F1F1F),
    .DRPREADYS31  (1'b1       )
);
```

```
drpChannelPort drpChannelPort0(
    .DRPCLK           (DRPCLK0),
    .DRPRESETn        (DRPRESET0n),
    .DRPADDR          (DRPADDR0),
    .DRPSEL           (DRPSEL0),
    .DRPWRITE         (DRPWRITE0),
    .DRPWDATA         (DRPWDATA0),
    .DRPRDATA         (DRPRDATA0),
    .DRPREADY         (DRPREADY0),

    .X_DRPCLK         (X_DRPCLK0),
    .X_DRPRESETn      (X_DRPRESET0n),
    .X_DRPADDR        (X_DRPADDR0),
    .X_DRPSEL         (X_DRPSEL0),
    .X_DRPWRITE       (X_DRPWRITE0),
    .X_DRPWDATA       (X_DRPWDATA0),
    .X_DRPRDATA       (X_DRPRDATA0),
    .X_DRPREADY       (X_DRPREADY0),

    .asRstN           (asRstN),

    .asBroadcast      (asBroadcast0),
    .asGpoMuxSel      (asGpoMuxSel0),
    .asSoftRst        (asSoftRst0),
    .asLoopback       (asLoopback0),
    .asTxSysClkSel    (asTxSysClkSel0),
    .asRxSysClkSel    (asRxSysClkSel0),
    .asTxOutClkSel    (asTxOutClkSel0),
    .asRxOutClkSel    (asRxOutClkSel0),
    .asTxDiffCtrl     (asTxDiffCtrl0),
    .asTxMainCursor   (asTxMainCursor0),
    .asTxPreCursor    (asTxPreCursor0),
    .asTxPostCursor   (asTxPostCursor0),

    .txUsrClk2        (txUsrClk2[0]),
    .txUsrRstN        (txUsrRstN_SYNC[0]),
    .txRdy            (gt0_qpllllock_i),

    .txData           (txData0),
    .txPrbsSel        (txPrbsSel0),
    .txpd             (txpdin0),
    .txPrbsForceErrLvl (txPrbsForceErrLvl0),
    .txPrbsForceErrPls (txPrbsForceErrPls0),
    .txPolarity       (txPolarity0),
    .txResetDone      (txResetDone0),
    .tx8b10bEn        (tx8b10bEn0),
    .txRate           (txRate0),

    .rxUsrClk2        (rxUsrClk2[0]),
    .rxUsrRstN        (rxUsrRstN_SYNC[0]),
    .rxRdy            (gt0_qpllllock_i),

    .rxData           (rxData0),
    .rxPrbsSel        (rxPrbsSel0),
    .rxPrbsCntRstPls  (rxPrbsCntRstPls0),
    .rxPolarity       (rxPolarity0),
    .rxPrbsErrFlag    (rxPrbsErrFlag0),
    .rxResetDone      (rxResetDone0),
    .rx8b10bEn        (rx8b10bEn0),
    .rxRate           (rxRate0)
);

drpChannelPort drpChannelPort1(
    .DRPCLK           (DRPCLK1),
    .DRPRESETn        (DRPRESET1n),
    .DRPADDR          (DRPADDR1),
    .DRPSEL           (DRPSEL1),
    .DRPWRITE         (DRPWRITE1),
    .DRPWDATA         (DRPWDATA1),
    .DRPRDATA         (DRPRDATA1),
```

```
    .DRPREADY         (DRPREADY1),

    .X_DRPCLK         (X_DRPCLK1),
    .X_DRPRESETn      (X_DRPRESET1n),
    .X_DRPADDR        (X_DRPADDR1),
    .X_DRPSEL         (X_DRPSEL1),
    .X_DRPWRITE       (X_DRPWRITE1),
    .X_DRPWDATA       (X_DRPWDATA1),
    .X_DRPRDATA       (X_DRPRDATA1),
    .X_DRPREADY       (X_DRPREADY1),

    .asRstN           (asRstN),

    .asBroadcast      (), //unconnected - special register is only used in channel port 0
    .asGpoMuxSel      (), //unconnected - special register is only used in channel port 0
    .asSoftRst        (), //unconnected - special register is only used in channel port 0
    .asLoopback       (asLoopback1),
    .asTxSysClkSel    (asTxSysClkSel1),
    .asRxSysClkSel    (asRxSysClkSel1),
    .asTxOutClkSel    (asTxOutClkSel1),
    .asRxOutClkSel    (asRxOutClkSel1),
    .asTxDiffCtrl     (asTxDiffCtrl1),
    .asTxMainCursor   (asTxMainCursor1),
    .asTxPreCursor    (asTxPreCursor1),
    .asTxPostCursor   (asTxPostCursor1),

    .txUsrClk2        (txUsrClk2[1]),
    .txUsrRstN        (txUsrRstN_SYNC[1]),
    .txRdy            (gt1_qpllock_i),

    .txData           (txData1),
    .txPrbsSel        (txPrbsSel1),
    .txpd             (txpdin1),
    .txPrbsForceErrLvl (txPrbsForceErrLvl1),
    .txPrbsForceErrPls (txPrbsForceErrPls1),
    .txPolarity       (txPolarity1),
    .txResetDone      (txResetDone1),
    .tx8b10bEn        (tx8b10bEn1),
    .txRate           (txRate1),

    .rxUsrClk2        (rxUsrClk2[1]),
    .rxUsrRstN        (rxUsrRstN_SYNC[1]),
    .rxRdy            (gt1_qpllock_i),

    .rxData           (rxData1),
    .rxPrbsSel        (rxPrbsSel1),
    .rxPrbsCntRstPls  (rxPrbsCntRstPls1),
    .rxPolarity       (rxPolarity1),
    .rxPrbsErrFlag    (rxPrbsErrFlag1),
    .rxResetDone      (rxResetDone1),
    .rx8b10bEn        (rx8b10bEn1),
    .rxRate           (rxRate1)
);

drpChannelPort drpChannelPort2(
    .DRPCLK           (DRPCLK2),
    .DRPRESETn        (DRPRESET2n),
    .DRPADDR          (DRPADDR2),
    .DRPSEL           (DRPSEL2),
    .DRPWRITE         (DRPWRITE2),
    .DRPWDATA         (DRPWDATA2),
    .DRPRDATA         (DRPRDATA2),
    .DRPREADY         (DRPREADY2),

    .X_DRPCLK         (X_DRPCLK2),
    .X_DRPRESETn      (X_DRPRESET2n),
    .X_DRPADDR        (X_DRPADDR2),
    .X_DRPSEL         (X_DRPSEL2),
    .X_DRPWRITE       (X_DRPWRITE2),
    .X_DRPWDATA       (X_DRPWDATA2),
```

```
    .X_DRPRDATA       (X_DRPRDATA2),
    .X_DRPREADY       (X_DRPREADY2),

    .asRstN           (asRstN),

    .asBroadcast      (), //unconnected - special register is only used in channel port 0
    .asGpoMuxSel      (), //unconnected - special register is only used in channel port 0
    .asSoftRst        (), //unconnected - special register is only used in channel port 0
    .asLoopback       (asLoopback2),
    .asTxSysClkSel    (asTxSysClkSel2),
    .asRxSysClkSel    (asRxSysClkSel2),
    .asTxOutClkSel    (asTxOutClkSel2),
    .asRxOutClkSel    (asRxOutClkSel2),
    .asTxDiffCtrl     (asTxDiffCtrl2),
    .asTxMainCursor   (asTxMainCursor2),
    .asTxPreCursor    (asTxPreCursor2),
    .asTxPostCursor   (asTxPostCursor2),

    .txUsrClk2        (txUsrClk2[2]),
    .txUsrRstN        (txUsrRstN_SYNC[2]),
    .txRdy            (gt2_qpllock_i),

    .txData           (txData2),
    .txPrbsSel        (txPrbsSel2),
    .txpd             (txpdin2),
    .txPrbsForceErrLvl (txPrbsForceErrLvl2),
    .txPrbsForceErrPls (txPrbsForceErrPls2),
    .txPolarity       (txPolarity2),
    .txResetDone      (txResetDone2),
    .tx8b10bEn        (tx8b10bEn2),
    .txRate           (txRate2),

    .rxUsrClk2        (rxUsrClk2[2]),
    .rxUsrRstN        (rxUsrRstN_SYNC[2]),
    .rxRdy            (gt2_qpllock_i),

    .rxData           (rxData2),
    .rxPrbsSel        (rxPrbsSel2),
    .rxPrbsCntRstPls  (rxPrbsCntRstPls2),
    .rxPolarity       (rxPolarity2),
    .rxPrbsErrFlag    (rxPrbsErrFlag2),
    .rxResetDone      (rxResetDone2),
    .rx8b10bEn        (rx8b10bEn2),
    .rxRate           (rxRate2)
);

drpChannelPort drpChannelPort3(
    .DRPCLK           (DRPCLK3),
    .DRPRESETn        (DRPRESET3n),
    .DRPADDR          (DRPADDR3),
    .DRPSEL           (DRPSEL3),
    .DRPWRITE         (DRPWRITE3),
    .DRPWDATA         (DRPWDATA3),
    .DRPRDATA         (DRPRDATA3),
    .DRPREADY         (DRPREADY3),

    .X_DRPCLK         (X_DRPCLK3),
    .X_DRPRESETn      (X_DRPRESET3n),
    .X_DRPADDR        (X_DRPADDR3),
    .X_DRPSEL         (X_DRPSEL3),
    .X_DRPWRITE       (X_DRPWRITE3),
    .X_DRPWDATA       (X_DRPWDATA3),
    .X_DRPRDATA       (X_DRPRDATA3),
    .X_DRPREADY       (X_DRPREADY3),

    .asRstN           (asRstN),

    .asBroadcast      (), //unconnected - special register is only used in channel port 0
    .asGpoMuxSel      (), //unconnected - special register is only used in channel port 0
    .asSoftRst        (), //unconnected - special register is only used in channel port 0
```

```
    .asLoopback        (asLoopback3),
    .asTxSysClkSel     (asTxSysClkSel3),
    .asRxSysClkSel     (asRxSysClkSel3),
    .asTxOutClkSel     (asTxOutClkSel3),
    .asRxOutClkSel     (asRxOutClkSel3),
    .asTxDiffCtrl      (asTxDiffCtrl3),
    .asTxMainCursor    (asTxMainCursor3),
    .asTxPreCursor     (asTxPreCursor3),
    .asTxPostCursor    (asTxPostCursor3),

    .txUsrClk2         (txUsrClk2[3]),
    .txUsrRstN         (txUsrRstN_SYNC[3]),
    .txRdy             (gt3_qplllock_i),

    .txData            (txData3),
    .txPrbsSel         (txPrbsSel3),
    .txpd              (txpdin3),
    .txPrbsForceErrLvl (txPrbsForceErrLvl3),
    .txPrbsForceErrPls (txPrbsForceErrPls3),
    .txPolarity        (txPolarity3),
    .txResetDone       (txResetDone3),
    .tx8b10bEn         (tx8b10bEn3),
    .txRate            (txRate3),

    .rxUsrClk2         (rxUsrClk2[3]),
    .rxUsrRstN         (rxUsrRstN_SYNC[3]),
    .rxRdy             (gt3_qplllock_i),

    .rxData            (rxData3),
    .rxPrbsSel         (rxPrbsSel3),
    .rxPrbsCntRstPls   (rxPrbsCntRstPls3),
    .rxPolarity        (rxPolarity3),
    .rxPrbsErrFlag     (rxPrbsErrFlag3),
    .rxResetDone       (rxResetDone3),
    .rx8b10bEn         (rx8b10bEn3),
    .rxRate            (rxRate3)
);


// This drpChannelPort instantiation exists to enable commonication to a COMMON DRP Port
drpChannelPort drpChannelPort16(
    .DRPCLK            (DRPCLK16),
    .DRPRESETn         (DRPRESET16n),
    .DRPADDR           (DRPADDR16),
    .DRPSEL            (DRPSEL16),
    .DRPWRITE          (DRPWRITE16),
    .DRPWDATA          (DRPWDATA16),
    .DRPRDATA          (DRPRDATA16),
    .DRPREADY          (DRPREADY16),

    .X_DRPCLK          (X_DRPCLK16),
    .X_DRPRESETn       (X_DRPRESET16n),
    .X_DRPADDR         (X_DRPADDR16),
    .X_DRPSEL          (X_DRPSEL16),
    .X_DRPWRITE        (X_DRPWRITE16),
    .X_DRPWDATA        (X_DRPWDATA16),
    .X_DRPRDATA        (X_DRPRDATA16),
    .X_DRPREADY        (X_DRPREADY16),

    .asRstN            (asRstN),

    .asBroadcast       (),
    .asGpoMuxSel       (),
    .asSoftRst         (),
    .asLoopback        (),
    .asTxSysClkSel     (),
    .asRxSysClkSel     (),
    .asTxOutClkSel     (),
    .asRxOutClkSel     (),
    .asTxDiffCtrl      (),
```

```
    .asTxMainCursor    (),
    .asTxPreCursor     (),
    .asTxPostCursor    (),

    .txUsrClk2         (txUsrClk2[0]),
    .txUsrRstN         (txUsrRstN_SYNC[0]),
    .txRdy             (1'b1),

    .txData            (),
    .txPrbsSel         (),
    .txPrbsForceErrLvl (),
    .txPrbsForceErrPls (),
    .txPolarity        (),
    .txResetDone       (1'b0),
    .tx8b10bEn         (),
    .txRate            (),

    .rxUsrClk2         (rxUsrClk2[0]),
    .rxUsrRstN         (rxUsrRstN_SYNC[0]),
    .rxRdy             (1'b1),

    .rxData            (64'd0),
    .rxPrbsSel         (),
    .rxPrbsCntRstPls   (),
    .rxPolarity        (),
    .rxPrbsErrFlag     (1'b0),
    .rxResetDone       (1'b0),
    .rx8b10bEn         (),
    .rxRate            ()
);

// This drpChannelPort instantiation exists to enable commonication to a COMMON DRP Port
drpChannelPort drpChannelPort17(
    .DRPCLK            (DRPCLK17),
    .DRPRESETn         (DRPRESET17n),
    .DRPADDR           (DRPADDR17),
    .DRPSEL            (DRPSEL17),
    .DRPWRITE          (DRPWRITE17),
    .DRPWDATA          (DRPWDATA17),
    .DRPRDATA          (DRPRDATA17),
    .DRPREADY          (DRPREADY17),

    .X_DRPCLK          (X_DRPCLK17),
    .X_DRPRESETn       (X_DRPRESET17n),
    .X_DRPADDR         (X_DRPADDR17),
    .X_DRPSEL          (X_DRPSEL17),
    .X_DRPWRITE        (X_DRPWRITE17),
    .X_DRPWDATA        (X_DRPWDATA17),
    .X_DRPRDATA        (X_DRPRDATA17),
    .X_DRPREADY        (X_DRPREADY17),

    .asRstN            (asRstN),

    .asBroadcast       (),
    .asGpoMuxSel       (),
    .asSoftRst         (),
    .asLoopback        (),
    .asTxSysClkSel     (),
    .asRxSysClkSel     (),
    .asTxOutClkSel     (),
    .asRxOutClkSel     (),
    .asTxDiffCtrl      (),
    .asTxMainCursor    (),
    .asTxPreCursor     (),
    .asTxPostCursor    (),

    .txUsrClk2         (txUsrClk2[0]),
    .txUsrRstN         (txUsrRstN_SYNC[0]),
    .txRdy             (1'b1),
```

```
    .txData           (),
    .txPrbsSel        (),
    .txPrbsForceErrLvl (),
    .txPrbsForceErrPls (),
    .txPolarity       (),
    .txResetDone      (1'b0),
    .tx8b10bEn        (),
    .txRate           (),

    .rxUsrClk2        (rxUsrClk2[0]),
    .rxUsrRstN        (rxUsrRstN_SYNC[0]),
    .rxRdy            (1'b1),

    .rxData           (64'd0),
    .rxPrbsSel        (),
    .rxPrbsCntRstPls  (),
    .rxPolarity       (),
    .rxPrbsErrFlag    (1'b0),
    .rxResetDone      (1'b0),
    .rx8b10bEn        (),
    .rxRate           ()
  );



////////////////////////////////////////
// Xilinx IP Instantiations
////////////////////////////////////////

  gtx8_chan_pll_init #
  (
    .EXAMPLE_SIM_GTRESET_SPEEDUP    ("TRUE"),
    .STABLE_CLOCK_PERIOD            (20),
    .EXAMPLE_SIMULATION             (EXAMPLE_SIMULATION), //default of 0 is used for FPGA, value of 1 is passed down
from testbench level for simulation
    .EXAMPLE_USE_CHIPSCOPE          (0)
  )
  gtx8_chan_pll_init_i
  (
    .SYSCLK_IN                  (sysClk),                 //JEH ok
    .SOFT_RESET_IN              (~softRstN | asSoftRst0),   //JEH ok - soft reset can come from either a button push or as a result of
writing to the soft reset register
    .DONT_RESET_ON_DATA_ERROR_IN    (1'b0),                 //JEH ok
    .GT0_TX_FSM_RESET_DONE_OUT      (gt0_txfsmresetdone_i),     //JEH ok
    .GT0_RX_FSM_RESET_DONE_OUT      (gt0_rxfsmresetdone_i),     //JEH ok
    .GT0_DATA_VALID_IN              (1'b1),                 //JEH ok
    .GT1_TX_FSM_RESET_DONE_OUT      (gt1_txfsmresetdone_i),     //JEH ok
    .GT1_RX_FSM_RESET_DONE_OUT      (gt1_rxfsmresetdone_i),     //JEH ok
    .GT1_DATA_VALID_IN              (1'b1),                 //JEH ok
    .GT2_TX_FSM_RESET_DONE_OUT      (gt2_txfsmresetdone_i),     //JEH ok
    .GT2_RX_FSM_RESET_DONE_OUT      (gt2_rxfsmresetdone_i),     //JEH ok
    .GT2_DATA_VALID_IN              (1'b1),                 //JEH ok
    .GT3_TX_FSM_RESET_DONE_OUT      (gt3_txfsmresetdone_i),     //JEH ok
    .GT3_RX_FSM_RESET_DONE_OUT      (gt3_rxfsmresetdone_i),     //JEH ok
    .GT3_DATA_VALID_IN              (1'b1),                 //JEH ok


    //____________________________________________________________________________
    //____________________________________________________________________________
    //GT0  (X1Y12)
    .gt0_gtrefclk0_in               (usr_clk),
    //------------------------- Additional Ports ------------------------------
    .GT0_RXOUTCLKSEL_IN             (asRxOutClkSel0),
    .GT0_TXOUTCLKSEL_IN             (asTxOutClkSel0),
    .GT0_RXSYSCLKSEL_IN             (asRxSysClkSel0),
    .GT0_TXSYSCLKSEL_IN             (asTxSysClkSel0),
    .GT0_TX8B10BEN_IN               (tx8b10bEn0),
    .GT0_RX8B10BEN_IN               (rx8b10bEn0),
    .GT0_RXOUTCLK_OUT               (GT0_RXOUTCLK_OUT),
```

```
    //------------------------- Channel - DRP Ports  -------------------------
    .GT0_DRPADDR_IN               (X_DRPADDR0[11:3]),        //JEH ok
    .GT0_DRPCLK_IN                (X_DRPCLK0),               //JEH ok
    .GT0_DRPDI_IN                 (X_DRPWDATA0[15:0]),       //JEH ok
    .GT0_DRPDO_OUT                (X_DRPRDATA0[15:0]),       //JEH ok
    .GT0_DRPEN_IN                 (X_DRPSEL0),               //JEH ok
    .GT0_DRPRDY_OUT               (X_DRPREADY0),             //JEH ok
    .GT0_DRPWE_IN                 (X_DRPWRITE0),             //JEH ok
    //------------------------- Loopback Ports -----------------------------
    .GT0_LOOPBACK_IN              (asLoopback0),             //JEH ok
    //------------------------- PCI Express Ports -------------------------
    .GT0_RXRATE_IN                (rxRate0),                 //JEH ok
    //------------------ RX Initialization and Reset Ports --------------------
    .GT0_RXUSERRDY_IN             (1'b0),                    //JEH unused at this level - reset FSM inside the reset initialization handles
the timing of this.
    //------------------------ RX Margin Analysis Ports ------------------------
    .GT0_EYESCANDATAERROR_OUT     (),                        //JEH ok
    //---------------------- Receive Ports - CDR Ports -----------------------
    .GT0_RXCDRLOCK_OUT            (rxCdrLock0),              //JEH ok
    //---------------- Receive Ports - FPGA RX Interface Ports -----------------
    .GT0_RXUSRCLK_IN              (rxUsrClk[0]),             //JEH ok
    .GT0_RXUSRCLK2_IN             (rxUsrClk2[0]),            //JEH ok
    //---------------- Receive Ports - FPGA RX interface Ports -----------------
    .GT0_RXDATA_OUT               (rxData0[31:0]),           //JEH ok
    //---------------- Receive Ports - Pattern Checker Ports -----------------
    .GT0_RXPRBSERR_OUT            (rxPrbsErrFlag0),          //JEH ok
    .GT0_RXPRBSSEL_IN             (rxPrbsSel0),              //JEH ok
    //------------------ Receive Ports - Pattern Checker ports -----------------
    .GT0_RXPRBSCNTRESET_IN        (rxPrbsCntRstPls0),        //JEH ok
    //------------------------- Receive Ports - RX AFE -------------------------
    .GT0_GTXRXP_IN                (rxp[0]),                  //JEH ok
    //---------------------- Receive Ports - RX AFE Ports ----------------------
    .GT0_GTXRXN_IN                (rxn[0]),                  //JEH ok
    //---------- Receive Ports - RX Fabric ClocK Output Control Ports ----------
    .GT0_RXRATEDONE_OUT           (),                        //JEH ok (unused)
    //----------- Receive Ports - RX Initialization and Reset Ports ------------
    .GT0_GTRXRESET_IN             (!gt0_qplllock_i),         //JEH ok
    .GT0_RXPMARESET_IN            (1'b0),                    //JEH ok (example design drives this low)
    //---------------- Receive Ports - RX Polarity Control Ports -----------------
    .GT0_RXPOLARITY_IN            (rxPolarity0),             //JEH ok
    //-------------------- Receive Ports - RX gearbox ports --------------------
    .GT0_RXSLIDE_IN               (1'b0),                    //JEH ok - comma bump control unused
    //------------ Receive Ports -RX Initialization and Reset Ports ------------
    .GT0_RXRESETDONE_OUT          (rxResetDone0),            //JEH ok
    //---------------------- TX Configurable Driver Ports ----------------------
    .GT0_TXPOSTCURSOR_IN          (asTxPostCursor0),         //JEH ok
    .GT0_TXPRECURSOR_IN           (asTxPreCursor0),          //JEH ok
    //------------------- TX Initialization and Reset Ports --------------------
    .GT0_GTTXRESET_IN             (!gt0_qplllock_i),         //JEH ok
    .GT0_TXUSERRDY_IN             (1'b0),                    //JEH unused at this level - reset FSM inside the reset initialization handles
the timing of this.
    //---------------- Transmit Ports - FPGA TX Interface Ports ----------------
    .GT0_TXUSRCLK_IN              (txUsrClk[0]),             //JEH ok
    .GT0_TXUSRCLK2_IN             (txUsrClk2[0]),            //JEH ok
    //------------------- Transmit Ports - PCI Express Ports -------------------
    .GT0_TXRATE_IN                (txRate0),                 //JEH ok
    //---------------- Transmit Ports - Pattern Generator Ports ----------------
    .GT0_TXPRBSFORCEERR_IN        (txPrbsForceErrLvl0 | txPrbsForceErrPls0), //JEH ok
    //------------- Transmit Ports - TX Configurable Driver Ports --------------
    .GT0_TXDIFFCTRL_IN            (asTxDiffCtrl0),           //JEH ok
    .GT0_TXMAINCURSOR_IN          (asTxMainCursor0),         //JEH ok
    //---------------- Transmit Ports - TX Data Path interface -----------------
    .GT0_TXDATA_IN                (txData0[31:0]),           //JEH ok
    //-------------- Transmit Ports - TX Driver and OOB signaling --------------
    .GT0_GTXTXN_OUT               (txn[0]),                  //JEH ok
    .GT0_GTXTXP_OUT               (txp[0]),                  //JEH ok
    //--------- Transmit Ports - TX Fabric Clock Output Control Ports ----------
    .GT0_TXOUTCLK_OUT             (GT0_TXOUTCLK_OUT),        //JEH ok
    .GT0_TXOUTCLKFABRIC_OUT       (),                        //JEH ok (unused)
    .GT0_TXOUTCLKPCS_OUT          (),                        //JEH ok (unused)
```

```
    .GT0_TXRATEDONE_OUT           (),                     //JEH ok (unused)
    //---------- Transmit Ports - TX Initialization and Reset Ports ----------
    .GT0_TXRESETDONE_OUT          (txResetDone0),         //JEH ok
    //---------- Transmit Ports - TX Polarity Control Ports ----------
    .GT0_TXPOLARITY_IN            (txPolarity0),          //JEH ok
    //---------- Transmit Ports - pattern Generator Ports ----------
    .GT0_TXPRBSSEL_IN             (txPrbsSel0),           //JEH ok
    .GT0_TXPD_IN                  (txpdin0),


    //____________________________________________________________________
    //____________________________________________________________________
    //GT1  (X1Y13)
    .gt1_gtrefclk0_in             (usr_clk),
    //------------------------- Additional Ports ------------------------------
    .GT1_RXOUTCLKSEL_IN           (asRxOutClkSel1),
    .GT1_TXOUTCLKSEL_IN           (asTxOutClkSel1),
    .GT1_RXSYSCLKSEL_IN           (asRxSysClkSel1),
    .GT1_TXSYSCLKSEL_IN           (asTxSysClkSel1),
    .GT1_TX8B10BEN_IN             (tx8b10bEn1),
    .GT1_RX8B10BEN_IN             (rx8b10bEn1),
    .GT1_RXOUTCLK_OUT             (GT1_RXOUTCLK_OUT),

    //------------------------- Channel - DRP Ports  -------------------------
    .GT1_DRPADDR_IN               (X_DRPADDR1[11:3]),       //JEH ok
    .GT1_DRPCLK_IN                (X_DRPCLK1),              //JEH ok
    .GT1_DRPDI_IN                 (X_DRPWDATA1[15:0]),      //JEH ok
    .GT1_DRPDO_OUT                (X_DRPRDATA1[15:0]),      //JEH ok
    .GT1_DRPEN_IN                 (X_DRPSEL1),              //JEH ok
    .GT1_DRPRDY_OUT               (X_DRPREADY1),            //JEH ok
    .GT1_DRPWE_IN                 (X_DRPWRITE1),            //JEH ok
    //------------------------------ Loopback Ports -----------------------------
    .GT1_LOOPBACK_IN              (asLoopback1),            //JEH ok
    //-------------------------- PCI Express Ports ----------------------------
    .GT1_RXRATE_IN                (rxRate1),                //JEH ok
    //------------------ RX Initialization and Reset Ports --------------------
    .GT1_RXUSERRDY_IN             (1'b0),                   //JEH unused at this level - reset FSM inside the reset initialization handles
the timing of this.
    //------------------------ RX Margin Analysis Ports ------------------------
    .GT1_EYESCANDATAERROR_OUT     (),                       //JEH ok
    //---------------------- Receive Ports - CDR Ports -------------------------
    .GT1_RXCDRLOCK_OUT            (rxCdrLock1),             //JEH ok
    //---------------- Receive Ports - FPGA RX Interface Ports -----------------
    .GT1_RXUSRCLK_IN              (rxUsrClk[1]),            //JEH ok
    .GT1_RXUSRCLK2_IN             (rxUsrClk2[1]),           //JEH ok
    //---------------- Receive Ports - FPGA RX interface Ports -----------------
    .GT1_RXDATA_OUT               (rxData1[31:0]),          //JEH ok
    //----------------- Receive Ports - Pattern Checker Ports ------------------
    .GT1_RXPRBSERR_OUT            (rxPrbsErrFlag1),         //JEH ok
    .GT1_RXPRBSSEL_IN             (rxPrbsSel1),             //JEH ok
    //----------------- Receive Ports - Pattern Checker ports ------------------
    .GT1_RXPRBSCNTRESET_IN        (rxPrbsCntRstPls1),       //JEH ok
    //------------------------ Receive Ports - RX AFE ------------------------
    .GT1_GTXRXP_IN                (rxp[1]),                 //JEH ok
    //---------------------- Receive Ports - RX AFE Ports ----------------------
    .GT1_GTXRXN_IN                (rxn[1]),                 //JEH ok
    //---------- Receive Ports - RX Fabric ClocK Output Control Ports ----------
    .GT1_RXRATEDONE_OUT           (),                       //JEH ok (unused)
    //---------- Receive Ports - RX Initialization and Reset Ports ------------
    .GT1_GTRXRESET_IN             (!gt1_qplllock_i),        //JEH ok
    .GT1_RXPMARESET_IN            (1'b0),                   //JEH ok (example design drives this low)
    //---------------- Receive Ports - RX Polarity Control Ports ----------------
    .GT1_RXPOLARITY_IN            (rxPolarity1),            //JEH ok
    //--------------------- Receive Ports - RX gearbox ports ---------------------
    .GT1_RXSLIDE_IN               (1'b0),                   //JEH ok - comma bump control unused
    //------------ Receive Ports -RX Initialization and Reset Ports ------------
    .GT1_RXRESETDONE_OUT          (rxResetDone1),           //JEH ok
    //---------------------- TX Configurable Driver Ports ----------------------
    .GT1_TXPOSTCURSOR_IN          (asTxPostCursor1),        //JEH ok
    .GT1_TXPRECURSOR_IN           (asTxPreCursor1),         //JEH ok
```

```
    //------------------- TX Initialization and Reset Ports --------------------
    .GT1_GTTXRESET_IN              (!gt1_qplllock_i),         //JEH ok
    .GT1_TXUSERRDY_IN              (1'b0),                    //JEH unused at this level - reset FSM inside the reset initialization handles
the timing of this.
    //---------------- Transmit Ports - FPGA TX Interface Ports ----------------
    .GT1_TXUSRCLK_IN               (txUsrClk[1]),             //JEH ok
    .GT1_TXUSRCLK2_IN              (txUsrClk2[1]),            //JEH ok
    //------------------- Transmit Ports - PCI Express Ports -------------------
    .GT1_TXRATE_IN                 (txRate1),                 //JEH ok
    //---------------- Transmit Ports - Pattern Generator Ports ----------------
    .GT1_TXPRBSFORCEERR_IN         (txPrbsForceErrLvl1 | txPrbsForceErrPls1), //JEH ok
    //------------- Transmit Ports - TX Configurable Driver Ports --------------
    .GT1_TXDIFFCTRL_IN             (asTxDiffCtrl1),           //JEH ok
    .GT1_TXMAINCURSOR_IN           (asTxMainCursor1),         //JEH ok
    //---------------- Transmit Ports - TX Data Path interface -----------------
    .GT1_TXDATA_IN                 (txData1[31:0]),           //JEH ok
    //-------------- Transmit Ports - TX Driver and OOB signaling --------------
    .GT1_GTXTXN_OUT                (txn[1]),                  //JEH ok
    .GT1_GTXTXP_OUT                (txp[1]),                  //JEH ok
    //--------- Transmit Ports - TX Fabric Clock Output Control Ports ----------
    .GT1_TXOUTCLK_OUT              (GT1_TXOUTCLK_OUT),        //JEH ok
    .GT1_TXOUTCLKFABRIC_OUT        (),                        //JEH ok (unused)
    .GT1_TXOUTCLKPCS_OUT           (),                        //JEH ok (unused)
    .GT1_TXRATEDONE_OUT            (),                        //JEH ok (unused)
    //----------- Transmit Ports - TX Initialization and Reset Ports -----------
    .GT1_TXRESETDONE_OUT           (txResetDone1),            //JEH ok
    //---------------- Transmit Ports - TX Polarity Control Ports ----------------
    .GT1_TXPOLARITY_IN             (txPolarity1),             //JEH ok
    //---------------- Transmit Ports - pattern Generator Ports ----------------
    .GT1_TXPRBSSEL_IN              (txPrbsSel1),              //JEH ok
    .GT1_TXPD_IN                   (txpdin1),


    //____________________________________________________________________________
    //____________________________________________________________________________
    //GT2  (X1Y14)
    .gt2_gtrefclk0_in              (usr_clk),
    //--------------------------- Additional Ports ------------------------------
    .GT2_RXOUTCLKSEL_IN            (asRxOutClkSel2),
    .GT2_TXOUTCLKSEL_IN            (asTxOutClkSel2),
    .GT2_RXSYSCLKSEL_IN            (asRxSysClkSel2),
    .GT2_TXSYSCLKSEL_IN            (asTxSysClkSel2),
    .GT2_TX8B10BEN_IN              (tx8b10bEn2),
    .GT2_RX8B10BEN_IN              (rx8b10bEn2),
    .GT2_RXOUTCLK_OUT              (GT2_RXOUTCLK_OUT),

    //--------------------------- Channel - DRP Ports  --------------------------
    .GT2_DRPADDR_IN                (X_DRPADDR2[11:3]),        //JEH ok
    .GT2_DRPCLK_IN                 (X_DRPCLK2),               //JEH ok
    .GT2_DRPDI_IN                  (X_DRPWDATA2[15:0]),       //JEH ok
    .GT2_DRPDO_OUT                 (X_DRPRDATA2[15:0]),       //JEH ok
    .GT2_DRPEN_IN                  (X_DRPSEL2),               //JEH ok
    .GT2_DRPRDY_OUT                (X_DRPREADY2),             //JEH ok
    .GT2_DRPWE_IN                  (X_DRPWRITE2),             //JEH ok
    //------------------------------ Loopback Ports -----------------------------
    .GT2_LOOPBACK_IN               (asLoopback2),             //JEH ok
    //---------------------------- PCI Express Ports ----------------------------
    .GT2_RXRATE_IN                 (rxRate2),                 //JEH ok
    //------------------- RX Initialization and Reset Ports --------------------
    .GT2_RXUSERRDY_IN              (1'b0),                    //JEH unused at this level - reset FSM inside the reset initialization handles
the timing of this.
    //------------------------ RX Margin Analysis Ports ------------------------
    .GT2_EYESCANDATAERROR_OUT      (),                        //JEH ok
    //------------------------ Receive Ports - CDR Ports -----------------------
    .GT2_RXCDRLOCK_OUT             (rxCdrLock2),              //JEH ok
    //---------------- Receive Ports - FPGA RX Interface Ports -----------------
    .GT2_RXUSRCLK_IN               (rxUsrClk[2]),             //JEH ok
    .GT2_RXUSRCLK2_IN              (rxUsrClk2[2]),            //JEH ok
    //---------------- Receive Ports - FPGA RX interface Ports -----------------
    .GT2_RXDATA_OUT                (rxData2[31:0]),           //JEH ok
```

```
    //------------------ Receive Ports - Pattern Checker Ports ------------------
    .GT2_RXPRBSERR_OUT            (rxPrbsErrFlag2),          //JEH ok
    .GT2_RXPRBSSEL_IN             (rxPrbsSel2),              //JEH ok
    //------------------ Receive Ports - Pattern Checker ports ------------------
    .GT2_RXPRBSCNTRESET_IN        (rxPrbsCntRstPls2),        //JEH ok
    //-------------------------- Receive Ports - RX AFE --------------------------
    .GT2_GTXRXP_IN                (rxp[2]),                  //JEH ok
    //----------------------- Receive Ports - RX AFE Ports -----------------------
    .GT2_GTXRXN_IN                (rxn[2]),                  //JEH ok
    //---------- Receive Ports - RX Fabric ClocK Output Control Ports ----------
    .GT2_RXRATEDONE_OUT           (),                        //JEH ok (unused)
    //------------ Receive Ports - RX Initialization and Reset Ports ------------
    .GT2_GTRXRESET_IN             (!gt2_qpllock_i),          //JEH ok
    .GT2_RXPMARESET_IN            (1'b0),                    //JEH ok (example design drives this low)
    //---------------- Receive Ports - RX Polarity Control Ports ----------------
    .GT2_RXPOLARITY_IN            (rxPolarity2),             //JEH ok
    //-------------------- Receive Ports - RX gearbox ports --------------------
    .GT2_RXSLIDE_IN               (1'b0),                    //JEH ok - comma bump control unused
    //------------ Receive Ports -RX Initialization and Reset Ports ------------
    .GT2_RXRESETDONE_OUT          (rxResetDone2),            //JEH ok
    //---------------------- TX Configurable Driver Ports ----------------------
    .GT2_TXPOSTCURSOR_IN          (asTxPostCursor2),         //JEH ok
    .GT2_TXPRECURSOR_IN           (asTxPreCursor2),          //JEH ok
    //------------------- TX Initialization and Reset Ports --------------------
    .GT2_GTTXRESET_IN             (!gt2_qpllock_i),          //JEH ok
    .GT2_TXUSERRDY_IN             (1'b0),                    //JEH unused at this level - reset FSM inside the reset initialization handles
the timing of this.
    //---------------- Transmit Ports - FPGA TX Interface Ports ----------------
    .GT2_TXUSRCLK_IN              (txUsrClk[2]),             //JEH ok
    .GT2_TXUSRCLK2_IN             (txUsrClk2[2]),            //JEH ok
    //------------------- Transmit Ports - PCI Express Ports -------------------
    .GT2_TXRATE_IN                (txRate2),                 //JEH ok
    //---------------- Transmit Ports - Pattern Generator Ports ----------------
    .GT2_TXPRBSFORCEERR_IN        (txPrbsForceErrLvl2 | txPrbsForceErrPls2), //JEH ok
    //------------- Transmit Ports - TX Configurable Driver Ports --------------
    .GT2_TXDIFFCTRL_IN            (asTxDiffCtrl2),           //JEH ok
    .GT2_TXMAINCURSOR_IN          (asTxMainCursor2),         //JEH ok
    //---------------- Transmit Ports - TX Data Path interface -----------------
    .GT2_TXDATA_IN                (txData2[31:0]),           //JEH ok
    //-------------- Transmit Ports - TX Driver and OOB signaling --------------
    .GT2_GTXTXN_OUT               (txn[2]),                  //JEH ok
    .GT2_GTXTXP_OUT               (txp[2]),                  //JEH ok
    //--------- Transmit Ports - TX Fabric Clock Output Control Ports ----------
    .GT2_TXOUTCLK_OUT             (GT2_TXOUTCLK_OUT),        //JEH ok
    .GT2_TXOUTCLKFABRIC_OUT       (),                        //JEH ok (unused)
    .GT2_TXOUTCLKPCS_OUT          (),                        //JEH ok (unused)
    .GT2_TXRATEDONE_OUT           (),                        //JEH ok (unused)
    //----------- Transmit Ports - TX Initialization and Reset Ports -----------
    .GT2_TXRESETDONE_OUT          (txResetDone2),            //JEH ok
    //---------------- Transmit Ports - TX Polarity Control Ports ----------------
    .GT2_TXPOLARITY_IN            (txPolarity2),             //JEH ok
    //----------------- Transmit Ports - pattern Generator Ports -----------------
    .GT2_TXPRBSSEL_IN             (txPrbsSel2),              //JEH ok
    .GT2_TXPD_IN                  (txpdin2),


    //____________________________________________________________________________
    //____________________________________________________________________________
    //GT3  (X1Y15)
    .gt3_gtrefclk0_in             (usr_clk),
    //-------------------------- Additional Ports ------------------------------
    .GT3_RXOUTCLKSEL_IN           (asRxOutClkSel3),
    .GT3_TXOUTCLKSEL_IN           (asTxOutClkSel3),
    .GT3_RXSYSCLKSEL_IN           (asRxSysClkSel3),
    .GT3_TXSYSCLKSEL_IN           (asTxSysClkSel3),
    .GT3_TX8B10BEN_IN             (tx8b10bEn3),
    .GT3_RX8B10BEN_IN             (rx8b10bEn3),
    .GT3_RXOUTCLK_OUT             (GT3_RXOUTCLK_OUT),

    //-------------------------- Channel - DRP Ports  --------------------------
```

```
    .GT3_DRPADDR_IN               (X_DRPADDR3[11:3]),        //JEH ok
    .GT3_DRPCLK_IN                (X_DRPCLK3),               //JEH ok
    .GT3_DRPDI_IN                 (X_DRPWDATA3[15:0]),       //JEH ok
    .GT3_DRPDO_OUT                (X_DRPRDATA3[15:0]),       //JEH ok
    .GT3_DRPEN_IN                 (X_DRPSEL3),               //JEH ok
    .GT3_DRPRDY_OUT               (X_DRPREADY3),             //JEH ok
    .GT3_DRPWE_IN                 (X_DRPWRITE3),             //JEH ok
    //------------------------------ Loopback Ports -----------------------------
    .GT3_LOOPBACK_IN              (asLoopback3),             //JEH ok
    //-------------------------- PCI Express Ports ----------------------------
    .GT3_RXRATE_IN                (rxRate3),                 //JEH ok
    //------------------- RX Initialization and Reset Ports --------------------
    .GT3_RXUSERRDY_IN             (1'b0),                    //JEH unused at this level - reset FSM inside the reset initialization handles
the timing of this.
    //------------------------ RX Margin Analysis Ports ------------------------
    .GT3_EYESCANDATAERROR_OUT     (),                        //JEH ok
    //----------------------- Receive Ports - CDR Ports ------------------------
    .GT3_RXCDRLOCK_OUT            (rxCdrLock3),              //JEH ok
    //----------------- Receive Ports - FPGA RX Interface Ports ----------------
    .GT3_RXUSRCLK_IN              (rxUsrClk[3]),             //JEH ok
    .GT3_RXUSRCLK2_IN             (rxUsrClk2[3]),            //JEH ok
    //----------------- Receive Ports - FPGA RX interface Ports ----------------
    .GT3_RXDATA_OUT               (rxData3[31:0]),           //JEH ok
    //----------------- Receive Ports - Pattern Checker Ports ------------------
    .GT3_RXPRBSERR_OUT            (rxPrbsErrFlag3),          //JEH ok
    .GT3_RXPRBSSEL_IN             (rxPrbsSel3),              //JEH ok
    //----------------- Receive Ports - Pattern Checker ports ------------------
    .GT3_RXPRBSCNTRESET_IN        (rxPrbsCntRstPls3),        //JEH ok
    //------------------------ Receive Ports - RX AFE --------------------------
    .GT3_GTXRXP_IN                (rxp[3]),                  //JEH ok
    //---------------------- Receive Ports - RX AFE Ports ----------------------
    .GT3_GTXRXN_IN                (rxn[3]),                  //JEH ok
    //---------- Receive Ports - RX Fabric ClocK Output Control Ports ----------
    .GT3_RXRATEDONE_OUT           (),                        //JEH ok (unused)
    //----------- Receive Ports - RX Initialization and Reset Ports ------------
    .GT3_GTRXRESET_IN             (!gt3_qplllock_i),         //JEH ok
    .GT3_RXPMARESET_IN            (1'b0),                    //JEH ok (example design drives this low)
    //---------------- Receive Ports - RX Polarity Control Ports ----------------
    .GT3_RXPOLARITY_IN            (rxPolarity3),             //JEH ok
    //--------------------- Receive Ports - RX gearbox ports ---------------------
    .GT3_RXSLIDE_IN               (1'b0),                    //JEH ok - comma bump control unused
    //------------ Receive Ports -RX Initialization and Reset Ports ------------
    .GT3_RXRESETDONE_OUT          (rxResetDone3),            //JEH ok
    //---------------------- TX Configurable Driver Ports ----------------------
    .GT3_TXPOSTCURSOR_IN          (asTxPostCursor3),         //JEH ok
    .GT3_TXPRECURSOR_IN           (asTxPreCursor3),          //JEH ok
    //------------------- TX Initialization and Reset Ports --------------------
    .GT3_GTTXRESET_IN             (!gt3_qplllock_i),         //JEH ok
    .GT3_TXUSERRDY_IN             (1'b0),                    //JEH unused at this level - reset FSM inside the reset initialization handles
the timing of this.
    //---------------- Transmit Ports - FPGA TX Interface Ports ----------------
    .GT3_TXUSRCLK_IN              (txUsrClk[3]),             //JEH ok
    .GT3_TXUSRCLK2_IN             (txUsrClk2[3]),            //JEH ok
    //------------------- Transmit Ports - PCI Express Ports -------------------
    .GT3_TXRATE_IN                (txRate3),                 //JEH ok
    //---------------- Transmit Ports - Pattern Generator Ports ----------------
    .GT3_TXPRBSFORCEERR_IN        (txPrbsForceErrLvl3 | txPrbsForceErrPls3), //JEH ok
    //------------- Transmit Ports - TX Configurable Driver Ports --------------
    .GT3_TXDIFFCTRL_IN            (asTxDiffCtrl3),           //JEH ok
    .GT3_TXMAINCURSOR_IN          (asTxMainCursor3),         //JEH ok
    //---------------- Transmit Ports - TX Data Path interface -----------------
    .GT3_TXDATA_IN                (txData3[31:0]),           //JEH ok
    //-------------- Transmit Ports - TX Driver and OOB signaling --------------
    .GT3_GTXTXN_OUT               (txn[3]),                  //JEH ok
    .GT3_GTXTXP_OUT               (txp[3]),                  //JEH ok
    //--------- Transmit Ports - TX Fabric Clock Output Control Ports ----------
    .GT3_TXOUTCLK_OUT             (GT3_TXOUTCLK_OUT),        //JEH ok
    .GT3_TXOUTCLKFABRIC_OUT       (),                        //JEH ok (unused)
    .GT3_TXOUTCLKPCS_OUT          (),                        //JEH ok (unused)
    .GT3_TXRATEDONE_OUT           (),                        //JEH ok (unused)
```

```
    //----------- Transmit Ports - TX Initialization and Reset Ports -----------
    .GT3_TXRESETDONE_OUT          (txResetDone3),           //JEH ok
    //---------------- Transmit Ports - TX Polarity Control Ports ----------------
    .GT3_TXPOLARITY_IN            (txPolarity3),            //JEH ok
    //---------------- Transmit Ports - pattern Generator Ports ----------------
    .GT3_TXPRBSSEL_IN             (txPrbsSel3),             //JEH ok
    .GT3_TXPD_IN                  (txpdin3),




  //____________________________________COMMON PORTS_____________________________________
    //--------------------- Common Block  - Ref Clock Ports ---------------------
    .GT0_GTREFCLK0_COMMON_IN      (usr_clk),                //JEH ok
    //----------------------- Common Block - QPLL Ports -----------------------
    .GT0_QPLLLOCK_OUT             (gt0_qpllock_i),          //JEH ok
    .GT0_QPLLLOCKDETCLK_IN        (X_DRPCLK0),              //JEH ok
    .GT0_QPLLRESET_IN             (1'b0),                   //JEH ok unused at this level - reset FSM inside the reset initialization handles
the timing of this.

    .GTC0_DRPADDR                 (X_DRPADDR16[10:3]),
    .GTC0_DRPCLK                  (X_DRPCLK16),
    .GTC0_DRPDI                   (X_DRPWDATA16[15:0]),
    .GTC0_DRPDO                   (X_DRPRDATA16[15:0]),
    .GTC0_DRPEN                   (X_DRPSEL16),
    .GTC0_DRPRDY                  (X_DRPREADY16),
    .GTC0_DRPWE                   (X_DRPWRITE16)

  );


  gtx8_chan_pll_GT_USRCLK_SOURCE gt_usrclk_source
  (
  .Q0_CLK1_GTREFCLK_PAD_N_IN  (usr_clk_n),
  .Q0_CLK1_GTREFCLK_PAD_P_IN  (usr_clk_p),
  .Q0_CLK1_GTREFCLK_OUT       (usr_clk),

  .GT0_TXUSRCLK_OUT    (txUsrClk[0]),
  .GT0_TXUSRCLK2_OUT   (txUsrClk2[0]),
  .GT0_TXOUTCLK_IN     (GT0_TXOUTCLK_OUT),
  .GT0_RXUSRCLK_OUT    (rxUsrClk[0]),
  .GT0_RXUSRCLK2_OUT   (rxUsrClk2[0]),

  .GT1_TXUSRCLK_OUT    (txUsrClk[1]),
  .GT1_TXUSRCLK2_OUT   (txUsrClk2[1]),
  .GT1_TXOUTCLK_IN     (GT1_TXOUTCLK_OUT),
  .GT1_RXUSRCLK_OUT    (rxUsrClk[1]),
  .GT1_RXUSRCLK2_OUT   (rxUsrClk2[1]),

  .GT2_TXUSRCLK_OUT    (txUsrClk[2]),
  .GT2_TXUSRCLK2_OUT   (txUsrClk2[2]),
  .GT2_TXOUTCLK_IN     (GT2_TXOUTCLK_OUT),
  .GT2_RXUSRCLK_OUT    (rxUsrClk[2]),
  .GT2_RXUSRCLK2_OUT   (rxUsrClk2[2]),

  .GT3_TXUSRCLK_OUT    (txUsrClk[3]),
  .GT3_TXUSRCLK2_OUT   (txUsrClk2[3]),
  .GT3_TXOUTCLK_IN     (GT3_TXOUTCLK_OUT),
  .GT3_RXUSRCLK_OUT    (rxUsrClk[3]),
  .GT3_RXUSRCLK2_OUT   (rxUsrClk2[3]),

  .DRPCLK_IN (1'b0),
  .DRPCLK_OUT()
);

BUFG rxoutclk_debug_bufg
(
  .I                    (GT1_RXOUTCLK_OUT),
```

```
   .O                          (GT0_RXOUTCLK_OUT_BUFG)
);

BUFG txoutclk_debug_bufg
(
   .I                         (GT3_TXOUTCLK_OUT),
   .O                          (GT0_TXOUTCLK_OUT_BUFG)
);


//////////////////////////////////////
// TX Reset Synchronizer Instantiations
//////////////////////////////////////

   socSync #(.SIGNAL_WIDTH(1), .INIT_VAL(0)) i_socSync_txUsrClk2RstN0_sync (
      .outClk     (txUsrClk2[0]),
      .outRstN    (txRstN),
      .asyncInput (txRstN),
      .syncOutput (txUsrRstN_SYNC[0])
   );

   socSync #(.SIGNAL_WIDTH(1), .INIT_VAL(0)) i_socSync_txUsrClk2RstN1_sync (
      .outClk     (txUsrClk2[1]),
      .outRstN    (txRstN),
      .asyncInput (txRstN),
      .syncOutput (txUsrRstN_SYNC[1])
   );

   socSync #(.SIGNAL_WIDTH(1), .INIT_VAL(0)) i_socSync_txUsrClk2RstN2_sync (
      .outClk     (txUsrClk2[2]),
      .outRstN    (txRstN),
      .asyncInput (txRstN),
      .syncOutput (txUsrRstN_SYNC[2])
   );

   socSync #(.SIGNAL_WIDTH(1), .INIT_VAL(0)) i_socSync_txUsrClk2RstN3_sync (
      .outClk     (txUsrClk2[3]),
      .outRstN    (txRstN),
      .asyncInput (txRstN),
      .syncOutput (txUsrRstN_SYNC[3])
   );


//////////////////////////////////////
// RX Reset Synchronizer Instantiations
//////////////////////////////////////

   socSync #(.SIGNAL_WIDTH(1), .INIT_VAL(0)) i_socSync_rxUsrClk2RstN0_sync (
      .outClk     (rxUsrClk2[0]),
      .outRstN    (rxRstN),
      .asyncInput (rxRstN),
      .syncOutput (rxUsrRstN_SYNC[0])
   );

   socSync #(.SIGNAL_WIDTH(1), .INIT_VAL(0)) i_socSync_rxUsrClk2RstN1_sync (
      .outClk     (rxUsrClk2[1]),
      .outRstN    (rxRstN),
      .asyncInput (rxRstN),
      .syncOutput (rxUsrRstN_SYNC[1])
   );

   socSync #(.SIGNAL_WIDTH(1), .INIT_VAL(0)) i_socSync_rxUsrClk2RstN2_sync (
      .outClk     (rxUsrClk2[2]),
      .outRstN    (rxRstN),
      .asyncInput (rxRstN),
      .syncOutput (rxUsrRstN_SYNC[2])
   );

   socSync #(.SIGNAL_WIDTH(1), .INIT_VAL(0)) i_socSync_rxUsrClk2RstN3_sync (
      .outClk     (rxUsrClk2[3]),
```

```
    .outRstN    (rxRstN),
    .asyncInput (rxRstN),
    .syncOutput (rxUsrRstN_SYNC[3])
  );

Endmodule

module socPxi (
  //reset, clock
  rstN,
  clk,

  //uart signals
  sin,
  sout,
  uart_divisor,

  //processor interface signals
  procXferEn,
  procXferDone,
  procWaitN,
  procAddr,
  procDin,
  procDout,
  procWrRdN,
  procDsize,

  //debug signals
  pxiDout,
  pxiDin,
  testvector
  );

  input rstN;
  input clk;

  //UART signals
  input  sin;               // serial data in
  output sout;              // serial data out
  input  [15:0] uart_divisor;

  // signals to/from system:
  output procXferEn;
  output procXferDone;          // high at end of bus cycle (1 clock)
  output [31:0] procAddr;       // address from processor
  input  [63:0] procDin;        // data from system to processor
  output [63:0] procDout;       // data from processor to system
  output procWrRdN;             // write read/not (1=write 0=read)
  input  procWaitN;             // active low wait state request from system
  output [1:0] procDsize;       // data size (8/16/32 bit)

  // debug ports:
  output [63:0] pxiDout;        // data from processor extension interface to here
  output [63:0] pxiDin;         // data from here to processor extension interface
  output [43:0] testvector;

  wire [31:0] pxiAddr;       // address from processor extension interface
  wire  [7:0] pxiCtrl;       // processor extension interface control word
  wire pxiActive;

  socUartIf socUartIf (
    .rstN         (rstN),
    .clk          (clk),
    .sin          (sin),
    .sout         (sout),

    .pxiAddr      (pxiAddr),
    .pxiDout      (pxiDout),
    .pxiDin       (pxiDin),
    .pxiCtrl      (pxiCtrl),
```

```
    .pxiActive      (pxiActive),
    .procXferDone   (procXferDone),
    .uart_divisor   (uart_divisor),
    .testvector     (testvector)
    );

// Processor Emulator:
socProcEmul socProcEmul (
    .pxiRstN        (rstN),
    .pxiClk         (clk),
    .pxiAddr        (pxiAddr),
    .pxiDout        (pxiDout),
    .pxiDin         (pxiDin),
    .pxiCtrl        (pxiCtrl),
    .pxiActive      (pxiActive),

    .procXferEn     (procXferEn),
    .procXferDone   (procXferDone),
    .procWaitN      (procWaitN),
    .procAddr       (procAddr),
    .procDin        (procDin),
    .procDout       (procDout),
    .procWrRdN      (procWrRdN),
    .procDsize      (procDsize)
);


module socDrpChannel (
    DRPCLK,
    DRPRESETn,
    DRP_Broadcast,

    DRPCLKM,
    DRPRESETMn,
    DRPADDRM,
    DRPSELM,
    DRPWRITEM,
    DRPWDATAM,
    DRPRDATAM,
    DRPREADYM,

    DRPCLKS0,
    DRPRESETS0n,
    DRPADDRS0,
    DRPSELS0,
    DRPWRITES0,
    DRPWDATAS0,
    DRPRDATAS0,
    DRPREADYS0,

    DRPCLKS1,
    DRPRESETS1n,
    DRPADDRS1,
    DRPSELS1,
    DRPWRITES1,
    DRPWDATAS1,
    DRPRDATAS1,
    DRPREADYS1,

    DRPCLKS2,
    DRPRESETS2n,
    DRPADDRS2,
    DRPSELS2,
    DRPWRITES2,
    DRPWDATAS2,
    DRPRDATAS2,
    DRPREADYS2,

    DRPCLKS3,
    DRPRESETS3n,
```

DRPADDRS3,
DRPSELS3,
DRPWRITES3,
DRPWDATAS3,
DRPRDATAS3,
DRPREADYS3,

DRPCLKS4,
DRPRESETS4n,
DRPADDRS4,
DRPSELS4,
DRPWRITES4,
DRPWDATAS4,
DRPRDATAS4,
DRPREADYS4,

DRPCLKS5,
DRPRESETS5n,
DRPADDRS5,
DRPSELS5,
DRPWRITES5,
DRPWDATAS5,
DRPRDATAS5,
DRPREADYS5,

DRPCLKS6,
DRPRESETS6n,
DRPADDRS6,
DRPSELS6,
DRPWRITES6,
DRPWDATAS6,
DRPRDATAS6,
DRPREADYS6,

DRPCLKS7,
DRPRESETS7n,
DRPADDRS7,
DRPSELS7,
DRPWRITES7,
DRPWDATAS7,
DRPRDATAS7,
DRPREADYS7,

DRPCLKS8,
DRPRESETS8n,
DRPADDRS8,
DRPSELS8,
DRPWRITES8,
DRPWDATAS8,
DRPRDATAS8,
DRPREADYS8,

DRPCLKS9,
DRPRESETS9n,
DRPADDRS9,
DRPSELS9,
DRPWRITES9,
DRPWDATAS9,
DRPRDATAS9,
DRPREADYS9,

DRPCLKS10,
DRPRESETS10n,
DRPADDRS10,
DRPSELS10,
DRPWRITES10,
DRPWDATAS10,
DRPRDATAS10,
DRPREADYS10,

DRPCLKS11,
DRPRESETS11n,
DRPADDRS11,
DRPSELS11,
DRPWRITES11,
DRPWDATAS11,
DRPRDATAS11,
DRPREADYS11,

DRPCLKS12,
DRPRESETS12n,
DRPADDRS12,
DRPSELS12,
DRPWRITES12,
DRPWDATAS12,
DRPRDATAS12,
DRPREADYS12,

DRPCLKS13,
DRPRESETS13n,
DRPADDRS13,
DRPSELS13,
DRPWRITES13,
DRPWDATAS13,
DRPRDATAS13,
DRPREADYS13,

DRPCLKS14,
DRPRESETS14n,
DRPADDRS14,
DRPSELS14,
DRPWRITES14,
DRPWDATAS14,
DRPRDATAS14,
DRPREADYS14,

DRPCLKS15,
DRPRESETS15n,
DRPADDRS15,
DRPSELS15,
DRPWRITES15,
DRPWDATAS15,
DRPRDATAS15,
DRPREADYS15,

DRPCLKS16,
DRPRESETS16n,
DRPADDRS16,
DRPSELS16,
DRPWRITES16,
DRPWDATAS16,
DRPRDATAS16,
DRPREADYS16,

DRPCLKS17,
DRPRESETS17n,
DRPADDRS17,
DRPSELS17,
DRPWRITES17,
DRPWDATAS17,
DRPRDATAS17,
DRPREADYS17,

DRPCLKS18,
DRPRESETS18n,
DRPADDRS18,
DRPSELS18,
DRPWRITES18,
DRPWDATAS18,
DRPRDATAS18,

DRPREADYS18,

DRPCLKS19,
DRPRESETS19n,
DRPADDRS19,
DRPSELS19,
DRPWRITES19,
DRPWDATAS19,
DRPRDATAS19,
DRPREADYS19,

DRPCLKS20,
DRPRESETS20n,
DRPADDRS20,
DRPSELS20,
DRPWRITES20,
DRPWDATAS20,
DRPRDATAS20,
DRPREADYS20,

DRPCLKS21,
DRPRESETS21n,
DRPADDRS21,
DRPSELS21,
DRPWRITES21,
DRPWDATAS21,
DRPRDATAS21,
DRPREADYS21,

DRPCLKS22,
DRPRESETS22n,
DRPADDRS22,
DRPSELS22,
DRPWRITES22,
DRPWDATAS22,
DRPRDATAS22,
DRPREADYS22,

DRPCLKS23,
DRPRESETS23n,
DRPADDRS23,
DRPSELS23,
DRPWRITES23,
DRPWDATAS23,
DRPRDATAS23,
DRPREADYS23,

DRPCLKS24,
DRPRESETS24n,
DRPADDRS24,
DRPSELS24,
DRPWRITES24,
DRPWDATAS24,
DRPRDATAS24,
DRPREADYS24,

DRPCLKS25,
DRPRESETS25n,
DRPADDRS25,
DRPSELS25,
DRPWRITES25,
DRPWDATAS25,
DRPRDATAS25,
DRPREADYS25,

DRPCLKS26,
DRPRESETS26n,
DRPADDRS26,
DRPSELS26,
DRPWRITES26,

```
DRPWDATAS26,
DRPRDATAS26,
DRPREADYS26,

DRPCLKS27,
DRPRESETS27n,
DRPADDRS27,
DRPSELS27,
DRPWRITES27,
DRPWDATAS27,
DRPRDATAS27,
DRPREADYS27,

DRPCLKS28,
DRPRESETS28n,
DRPADDRS28,
DRPSELS28,
DRPWRITES28,
DRPWDATAS28,
DRPRDATAS28,
DRPREADYS28,

DRPCLKS29,
DRPRESETS29n,
DRPADDRS29,
DRPSELS29,
DRPWRITES29,
DRPWDATAS29,
DRPRDATAS29,
DRPREADYS29,

DRPCLKS30,
DRPRESETS30n,
DRPADDRS30,
DRPSELS30,
DRPWRITES30,
DRPWDATAS30,
DRPRDATAS30,
DRPREADYS30,

DRPCLKS31,
DRPRESETS31n,
DRPADDRS31,
DRPSELS31,
DRPWRITES31,
DRPWDATAS31,
DRPRDATAS31,
DRPREADYS31
);

input  DRPCLK;
input  DRPRESETn;
input  [31:0] DRP_Broadcast;   //Special input from channel port 0 for broadcasting writes to multiple channels simultaneously

// DRP Mirrored Master interface
output       DRPCLKM;
output       DRPRESETMn;
input  [31:0] DRPADDRM;       // DRP Address
input        DRPSELM;        // DRP Peripheral Select
input        DRPWRITEM;      // DRP Write Strobe
input  [63:0] DRPWDATAM;      // DRP Write Data bus
output [63:0] DRPRDATAM;      // DRP Read Data bus Muxed output
output       DRPREADYM;      // DRP Ready Flag

// DRP Mirrored Slave interface 0
output       DRPCLKS0;
output       DRPRESETS0n;
output [31:0] DRPADDRS0;      // DRP Address
output       DRPSELS0;       // DRP Peripheral Select
output       DRPWRITES0;     // DRP Write Strobe
```

```
output [63:0] DRPWDATAS0;      // DRP Write Data bus
input  [63:0] DRPRDATAS0;      // DRP Read Data bus Muxed output
input         DRPREADYS0;      // DRP Ready Flag

// DRP Mirrored Slave interface 1
output        DRPCLKS1;
output        DRPRESETS1n;
output [31:0] DRPADDRS1;       // DRP Address
output        DRPSELS1;        // DRP Peripheral Select
output        DRPWRITES1;      // DRP Write Strobe
output [63:0] DRPWDATAS1;      // DRP Write Data bus
input  [63:0] DRPRDATAS1;      // DRP Read Data bus Muxed output
input         DRPREADYS1;      // DRP Ready Flag

// DRP Mirrored Slave interface 2
output        DRPCLKS2;
output        DRPRESETS2n;
output [31:0] DRPADDRS2;       // DRP Address
output        DRPSELS2;        // DRP Peripheral Select
output        DRPWRITES2;      // DRP Write Strobe
output [63:0] DRPWDATAS2;      // DRP Write Data bus
input  [63:0] DRPRDATAS2;      // DRP Read Data bus Muxed output
input         DRPREADYS2;      // DRP Ready Flag

// DRP Mirrored Slave interface 3
output        DRPCLKS3;
output        DRPRESETS3n;
output [31:0] DRPADDRS3;       // DRP Address
output        DRPSELS3;        // DRP Peripheral Select
output        DRPWRITES3;      // DRP Write Strobe
output [63:0] DRPWDATAS3;      // DRP Write Data bus
input  [63:0] DRPRDATAS3;      // DRP Read Data bus Muxed output
input         DRPREADYS3;      // DRP Ready Flag

// DRP Mirrored Slave interface 4
output        DRPCLKS4;
output        DRPRESETS4n;
output [31:0] DRPADDRS4;       // DRP Address
output        DRPSELS4;        // DRP Peripheral Select
output        DRPWRITES4;      // DRP Write Strobe
output [63:0] DRPWDATAS4;      // DRP Write Data bus
input  [63:0] DRPRDATAS4;      // DRP Read Data bus Muxed output
input         DRPREADYS4;      // DRP Ready Flag

// DRP Mirrored Slave interface 5
output        DRPCLKS5;
output        DRPRESETS5n;
output [31:0] DRPADDRS5;       // DRP Address
output        DRPSELS5;        // DRP Peripheral Select
output        DRPWRITES5;      // DRP Write Strobe
output [63:0] DRPWDATAS5;      // DRP Write Data bus
input  [63:0] DRPRDATAS5;      // DRP Read Data bus Muxed output
input         DRPREADYS5;      // DRP Ready Flag

// DRP Mirrored Slave interface 6
output        DRPCLKS6;
output        DRPRESETS6n;
output [31:0] DRPADDRS6;       // DRP Address
output        DRPSELS6;        // DRP Peripheral Select
output        DRPWRITES6;      // DRP Write Strobe
output [63:0] DRPWDATAS6;      // DRP Write Data bus
input  [63:0] DRPRDATAS6;      // DRP Read Data bus Muxed output
input         DRPREADYS6;      // DRP Ready Flag

// DRP Mirrored Slave interface 7
output        DRPCLKS7;
output        DRPRESETS7n;
output [31:0] DRPADDRS7;       // DRP Address
output        DRPSELS7;        // DRP Peripheral Select
output        DRPWRITES7;      // DRP Write Strobe
```

```
output [63:0] DRPWDATAS7;      // DRP Write Data bus
input  [63:0] DRPRDATAS7;      // DRP Read Data bus Muxed output
input         DRPREADYS7;      // DRP Ready Flag

// DRP Mirrored Slave interface 8
output        DRPCLKS8;
output        DRPRESETS8n;
output [31:0] DRPADDRS8;       // DRP Address
output        DRPSELS8;        // DRP Peripheral Select
output        DRPWRITES8;      // DRP Write Strobe
output [63:0] DRPWDATAS8;      // DRP Write Data bus
input  [63:0] DRPRDATAS8;      // DRP Read Data bus Muxed output
input         DRPREADYS8;      // DRP Ready Flag

// DRP Mirrored Slave interface 9
output        DRPCLKS9;
output        DRPRESETS9n;
output [31:0] DRPADDRS9;       // DRP Address
output        DRPSELS9;        // DRP Peripheral Select
output        DRPWRITES9;      // DRP Write Strobe
output [63:0] DRPWDATAS9;      // DRP Write Data bus
input  [63:0] DRPRDATAS9;      // DRP Read Data bus Muxed output
input         DRPREADYS9;      // DRP Ready Flag

// DRP Mirrored Slave interface 10
output        DRPCLKS10;
output        DRPRESETS10n;
output [31:0] DRPADDRS10;      // DRP Address
output        DRPSELS10;       // DRP Peripheral Select
output        DRPWRITES10;     // DRP Write Strobe
output [63:0] DRPWDATAS10;     // DRP Write Data bus
input  [63:0] DRPRDATAS10;     // DRP Read Data bus Muxed output
input         DRPREADYS10;     // DRP Ready Flag

// DRP Mirrored Slave interface 11
output        DRPCLKS11;
output        DRPRESETS11n;
output [31:0] DRPADDRS11;      // DRP Address
output        DRPSELS11;       // DRP Peripheral Select
output        DRPWRITES11;     // DRP Write Strobe
output [63:0] DRPWDATAS11;     // DRP Write Data bus
input  [63:0] DRPRDATAS11;     // DRP Read Data bus Muxed output
input         DRPREADYS11;     // DRP Ready Flag

// DRP Mirrored Slave interface 12
output        DRPCLKS12;
output        DRPRESETS12n;
output [31:0] DRPADDRS12;      // DRP Address
output        DRPSELS12;       // DRP Peripheral Select
output        DRPWRITES12;     // DRP Write Strobe
output [63:0] DRPWDATAS12;     // DRP Write Data bus
input  [63:0] DRPRDATAS12;     // DRP Read Data bus Muxed output
input         DRPREADYS12;     // DRP Ready Flag

// DRP Mirrored Slave interface 13
output        DRPCLKS13;
output        DRPRESETS13n;
output [31:0] DRPADDRS13;      // DRP Address
output        DRPSELS13;       // DRP Peripheral Select
output        DRPWRITES13;     // DRP Write Strobe
output [63:0] DRPWDATAS13;     // DRP Write Data bus
input  [63:0] DRPRDATAS13;     // DRP Read Data bus Muxed output
input         DRPREADYS13;     // DRP Ready Flag

// DRP Mirrored Slave interface 14
output        DRPCLKS14;
output        DRPRESETS14n;
output [31:0] DRPADDRS14;      // DRP Address
output        DRPSELS14;       // DRP Peripheral Select
output        DRPWRITES14;     // DRP Write Strobe
```

```
output [63:0] DRPWDATAS14;      // DRP Write Data bus
input  [63:0] DRPRDATAS14;      // DRP Read Data bus Muxed output
input         DRPREADYS14;      // DRP Ready Flag

// DRP Mirrored Slave interface 15
output        DRPCLKS15;
output        DRPRESETS15n;
output [31:0] DRPADDRS15;       // DRP Address
output        DRPSELS15;        // DRP Peripheral Select
output        DRPWRITES15;      // DRP Write Strobe
output [63:0] DRPWDATAS15;      // DRP Write Data bus
input  [63:0] DRPRDATAS15;      // DRP Read Data bus Muxed output
input         DRPREADYS15;      // DRP Ready Flag

// DRP Mirrored Slave interface 16
output        DRPCLKS16;
output        DRPRESETS16n;
output [31:0] DRPADDRS16;       // DRP Address
output        DRPSELS16;        // DRP Peripheral Select
output        DRPWRITES16;      // DRP Write Strobe
output [63:0] DRPWDATAS16;      // DRP Write Data bus
input  [63:0] DRPRDATAS16;      // DRP Read Data bus Muxed output
input         DRPREADYS16;      // DRP Ready Flag

// DRP Mirrored Slave interface 17
output        DRPCLKS17;
output        DRPRESETS17n;
output [31:0] DRPADDRS17;       // DRP Address
output        DRPSELS17;        // DRP Peripheral Select
output        DRPWRITES17;      // DRP Write Strobe
output [63:0] DRPWDATAS17;      // DRP Write Data bus
input  [63:0] DRPRDATAS17;      // DRP Read Data bus Muxed output
input         DRPREADYS17;      // DRP Ready Flag

// DRP Mirrored Slave interface 18
output        DRPCLKS18;
output        DRPRESETS18n;
output [31:0] DRPADDRS18;       // DRP Address
output        DRPSELS18;        // DRP Peripheral Select
output        DRPWRITES18;      // DRP Write Strobe
output [63:0] DRPWDATAS18;      // DRP Write Data bus
input  [63:0] DRPRDATAS18;      // DRP Read Data bus Muxed output
input         DRPREADYS18;      // DRP Ready Flag

// DRP Mirrored Slave interface 19
output        DRPCLKS19;
output        DRPRESETS19n;
output [31:0] DRPADDRS19;       // DRP Address
output        DRPSELS19;        // DRP Peripheral Select
output        DRPWRITES19;      // DRP Write Strobe
output [63:0] DRPWDATAS19;      // DRP Write Data bus
input  [63:0] DRPRDATAS19;      // DRP Read Data bus Muxed output
input         DRPREADYS19;      // DRP Ready Flag

// DRP Mirrored Slave interface 20
output        DRPCLKS20;
output        DRPRESETS20n;
output [31:0] DRPADDRS20;       // DRP Address
output        DRPSELS20;        // DRP Peripheral Select
output        DRPWRITES20;      // DRP Write Strobe
output [63:0] DRPWDATAS20;      // DRP Write Data bus
input  [63:0] DRPRDATAS20;      // DRP Read Data bus Muxed output
input         DRPREADYS20;      // DRP Ready Flag

// DRP Mirrored Slave interface 21
output        DRPCLKS21;
output        DRPRESETS21n;
output [31:0] DRPADDRS21;       // DRP Address
output        DRPSELS21;        // DRP Peripheral Select
output        DRPWRITES21;      // DRP Write Strobe
```

```
output [63:0] DRPWDATAS21;      // DRP Write Data bus
input  [63:0] DRPRDATAS21;      // DRP Read Data bus Muxed output
input         DRPREADYS21;      // DRP Ready Flag

// DRP Mirrored Slave interface 22
output        DRPCLKS22;
output        DRPRESETS22n;
output [31:0] DRPADDRS22;       // DRP Address
output        DRPSELS22;        // DRP Peripheral Select
output        DRPWRITES22;      // DRP Write Strobe
output [63:0] DRPWDATAS22;      // DRP Write Data bus
input  [63:0] DRPRDATAS22;      // DRP Read Data bus Muxed output
input         DRPREADYS22;      // DRP Ready Flag

// DRP Mirrored Slave interface 23
output        DRPCLKS23;
output        DRPRESETS23n;
output [31:0] DRPADDRS23;       // DRP Address
output        DRPSELS23;        // DRP Peripheral Select
output        DRPWRITES23;      // DRP Write Strobe
output [63:0] DRPWDATAS23;      // DRP Write Data bus
input  [63:0] DRPRDATAS23;      // DRP Read Data bus Muxed output
input         DRPREADYS23;      // DRP Ready Flag

// DRP Mirrored Slave interface 24
output        DRPCLKS24;
output        DRPRESETS24n;
output [31:0] DRPADDRS24;       // DRP Address
output        DRPSELS24;        // DRP Peripheral Select
output        DRPWRITES24;      // DRP Write Strobe
output [63:0] DRPWDATAS24;      // DRP Write Data bus
input  [63:0] DRPRDATAS24;      // DRP Read Data bus Muxed output
input         DRPREADYS24;      // DRP Ready Flag

// DRP Mirrored Slave interface 25
output        DRPCLKS25;
output        DRPRESETS25n;
output [31:0] DRPADDRS25;       // DRP Address
output        DRPSELS25;        // DRP Peripheral Select
output        DRPWRITES25;      // DRP Write Strobe
output [63:0] DRPWDATAS25;      // DRP Write Data bus
input  [63:0] DRPRDATAS25;      // DRP Read Data bus Muxed output
input         DRPREADYS25;      // DRP Ready Flag

// DRP Mirrored Slave interface 26
output        DRPCLKS26;
output        DRPRESETS26n;
output [31:0] DRPADDRS26;       // DRP Address
output        DRPSELS26;        // DRP Peripheral Select
output        DRPWRITES26;      // DRP Write Strobe
output [63:0] DRPWDATAS26;      // DRP Write Data bus
input  [63:0] DRPRDATAS26;      // DRP Read Data bus Muxed output
input         DRPREADYS26;      // DRP Ready Flag

// DRP Mirrored Slave interface 27
output        DRPCLKS27;
output        DRPRESETS27n;
output [31:0] DRPADDRS27;       // DRP Address
output        DRPSELS27;        // DRP Peripheral Select
output        DRPWRITES27;      // DRP Write Strobe
output [63:0] DRPWDATAS27;      // DRP Write Data bus
input  [63:0] DRPRDATAS27;      // DRP Read Data bus Muxed output
input         DRPREADYS27;      // DRP Ready Flag

// DRP Mirrored Slave interface 28
output        DRPCLKS28;
output        DRPRESETS28n;
output [31:0] DRPADDRS28;       // DRP Address
output        DRPSELS28;        // DRP Peripheral Select
output        DRPWRITES28;      // DRP Write Strobe
```

```
    output [63:0] DRPWDATAS28;      // DRP Write Data bus
    input  [63:0] DRPRDATAS28;      // DRP Read Data bus Muxed output
    input         DRPREADYS28;      // DRP Ready Flag

    // DRP Mirrored Slave interface 29
    output        DRPCLKS29;
    output        DRPRESETS29n;
    output [31:0] DRPADDRS29;       // DRP Address
    output        DRPSELS29;        // DRP Peripheral Select
    output        DRPWRITES29;      // DRP Write Strobe
    output [63:0] DRPWDATAS29;      // DRP Write Data bus
    input  [63:0] DRPRDATAS29;      // DRP Read Data bus Muxed output
    input         DRPREADYS29;      // DRP Ready Flag

    // DRP Mirrored Slave interface 30
    output        DRPCLKS30;
    output        DRPRESETS30n;
    output [31:0] DRPADDRS30;       // DRP Address
    output        DRPSELS30;        // DRP Peripheral Select
    output        DRPWRITES30;      // DRP Write Strobe
    output [63:0] DRPWDATAS30;      // DRP Write Data bus
    input  [63:0] DRPRDATAS30;      // DRP Read Data bus Muxed output
    input         DRPREADYS30;      // DRP Ready Flag

    // DRP Mirrored Slave interface 31
    output        DRPCLKS31;
    output        DRPRESETS31n;
    output [31:0] DRPADDRS31;       // DRP Address
    output        DRPSELS31;        // DRP Peripheral Select
    output        DRPWRITES31;      // DRP Write Strobe
    output [63:0] DRPWDATAS31;      // DRP Write Data bus
    input  [63:0] DRPRDATAS31;      // DRP Read Data bus Muxed output
    input         DRPREADYS31;      // DRP Ready Flag

// ---------------------------------
// Instantiations...
// ---------------------------------
wire [31:0] DRPSEL_Dec;

socDrpDec socDrpDec (
    .DRPADDR          (DRPADDRM),
    .DRPSEL_Dec       (DRPSEL_Dec)
    );

socDrpDataMux socDrpDataMux (
    .DRPSEL       (DRPSEL_Dec),
    .DRPRDATA_0   (DRPRDATAS0),
    .DRPRDATA_1   (DRPRDATAS1),
    .DRPRDATA_2   (DRPRDATAS2),
    .DRPRDATA_3   (DRPRDATAS3),
    .DRPRDATA_4   (DRPRDATAS4),
    .DRPRDATA_5   (DRPRDATAS5),
    .DRPRDATA_6   (DRPRDATAS6),
    .DRPRDATA_7   (DRPRDATAS7),
    .DRPRDATA_8   (DRPRDATAS8),
    .DRPRDATA_9   (DRPRDATAS9),
    .DRPRDATA_10  (DRPRDATAS10),
    .DRPRDATA_11  (DRPRDATAS11),
    .DRPRDATA_12  (DRPRDATAS12),
    .DRPRDATA_13  (DRPRDATAS13),
    .DRPRDATA_14  (DRPRDATAS14),
    .DRPRDATA_15  (DRPRDATAS15),
    .DRPRDATA_16  (DRPRDATAS16),
    .DRPRDATA_17  (DRPRDATAS17),
    .DRPRDATA_18  (DRPRDATAS18),
    .DRPRDATA_19  (DRPRDATAS19),
    .DRPRDATA_20  (DRPRDATAS20),
    .DRPRDATA_21  (DRPRDATAS21),
    .DRPRDATA_22  (DRPRDATAS22),
    .DRPRDATA_23  (DRPRDATAS23),
```

```
    .DRPRDATA_24  (DRPRDATAS24),
    .DRPRDATA_25  (DRPRDATAS25),
    .DRPRDATA_26  (DRPRDATAS26),
    .DRPRDATA_27  (DRPRDATAS27),
    .DRPRDATA_28  (DRPRDATAS28),
    .DRPRDATA_29  (DRPRDATAS29),
    .DRPRDATA_30  (DRPRDATAS30),
    .DRPRDATA_31  (DRPRDATAS31),
    .DRPRDATA     (DRPRDATAM),
    .DRPREADY_0   (DRPREADYS0),
    .DRPREADY_1   (DRPREADYS1),
    .DRPREADY_2   (DRPREADYS2),
    .DRPREADY_3   (DRPREADYS3),
    .DRPREADY_4   (DRPREADYS4),
    .DRPREADY_5   (DRPREADYS5),
    .DRPREADY_6   (DRPREADYS6),
    .DRPREADY_7   (DRPREADYS7),
    .DRPREADY_8   (DRPREADYS8),
    .DRPREADY_9   (DRPREADYS9),
    .DRPREADY_10  (DRPREADYS10),
    .DRPREADY_11  (DRPREADYS11),
    .DRPREADY_12  (DRPREADYS12),
    .DRPREADY_13  (DRPREADYS13),
    .DRPREADY_14  (DRPREADYS14),
    .DRPREADY_15  (DRPREADYS15),
    .DRPREADY_16  (DRPREADYS16),
    .DRPREADY_17  (DRPREADYS17),
    .DRPREADY_18  (DRPREADYS18),
    .DRPREADY_19  (DRPREADYS19),
    .DRPREADY_20  (DRPREADYS20),
    .DRPREADY_21  (DRPREADYS21),
    .DRPREADY_22  (DRPREADYS22),
    .DRPREADY_23  (DRPREADYS23),
    .DRPREADY_24  (DRPREADYS24),
    .DRPREADY_25  (DRPREADYS25),
    .DRPREADY_26  (DRPREADYS26),
    .DRPREADY_27  (DRPREADYS27),
    .DRPREADY_28  (DRPREADYS28),
    .DRPREADY_29  (DRPREADYS29),
    .DRPREADY_30  (DRPREADYS30),
    .DRPREADY_31  (DRPREADYS31),
    .DRPREADY     (DRPREADYM)
    );

// ---------------------------------
// Wired Aliases
// ---------------------------------
    assign DRPCLKM     = DRPCLK;
    assign DRPRESETMn  = DRPRESETn;

    // DRP Mirrored Slave interface 0
    assign DRPCLKS0    = DRPCLKM;
    assign DRPRESETS0n = DRPRESETMn;
    assign DRPADDRS0   = DRPADDRM;
    assign DRPSELS0    = DRPSELM & (DRPSEL_Dec[0] | (DRP_Broadcast[0] & DRPWRITEM));
    assign DRPWRITES0  = DRPWRITEM;
    assign DRPWDATAS0  = DRPWDATAM;

    // DRP Mirrored Slave interface 1
    assign DRPCLKS1    = DRPCLKM;
    assign DRPRESETS1n = DRPRESETMn;
    assign DRPADDRS1   = DRPADDRM;
    assign DRPSELS1    = DRPSELM & (DRPSEL_Dec[1] | (DRP_Broadcast[1] & DRPWRITEM));
    assign DRPWRITES1  = DRPWRITEM;
    assign DRPWDATAS1  = DRPWDATAM;

    // DRP Mirrored Slave interface 2
    assign DRPCLKS2    = DRPCLKM;
    assign DRPRESETS2n = DRPRESETMn;
    assign DRPADDRS2   = DRPADDRM;
```

```
assign DRPSELS2    = DRPSELM & (DRPSEL_Dec[2] | (DRP_Broadcast[2] & DRPWRITEM));
assign DRPWRITES2  = DRPWRITEM;
assign DRPWDATAS2  = DRPWDATAM;

// DRP Mirrored Slave interface 3
assign DRPCLKS3    = DRPCLKM;
assign DRPRESETS3n = DRPRESETMn;
assign DRPADDRS3   = DRPADDRM;
assign DRPSELS3    = DRPSELM & (DRPSEL_Dec[3] | (DRP_Broadcast[3] & DRPWRITEM));
assign DRPWRITES3  = DRPWRITEM;
assign DRPWDATAS3  = DRPWDATAM;

// DRP Mirrored Slave interface 4
assign DRPCLKS4    = DRPCLKM;
assign DRPRESETS4n = DRPRESETMn;
assign DRPADDRS4   = DRPADDRM;
assign DRPSELS4    = DRPSELM & (DRPSEL_Dec[4] | (DRP_Broadcast[4] & DRPWRITEM));
assign DRPWRITES4  = DRPWRITEM;
assign DRPWDATAS4  = DRPWDATAM;

// DRP Mirrored Slave interface 5
assign DRPCLKS5    = DRPCLKM;
assign DRPRESETS5n = DRPRESETMn;
assign DRPADDRS5   = DRPADDRM;
assign DRPSELS5    = DRPSELM & (DRPSEL_Dec[5] | (DRP_Broadcast[5] & DRPWRITEM));
assign DRPWRITES5  = DRPWRITEM;
assign DRPWDATAS5  = DRPWDATAM;

// DRP Mirrored Slave interface 6
assign DRPCLKS6    = DRPCLKM;
assign DRPRESETS6n = DRPRESETMn;
assign DRPADDRS6   = DRPADDRM;
assign DRPSELS6    = DRPSELM & (DRPSEL_Dec[6] | (DRP_Broadcast[6] & DRPWRITEM));
assign DRPWRITES6  = DRPWRITEM;
assign DRPWDATAS6  = DRPWDATAM;

// DRP Mirrored Slave interface 7
assign DRPCLKS7    = DRPCLKM;
assign DRPRESETS7n = DRPRESETMn;
assign DRPADDRS7   = DRPADDRM;
assign DRPSELS7    = DRPSELM & (DRPSEL_Dec[7] | (DRP_Broadcast[7] & DRPWRITEM));
assign DRPWRITES7  = DRPWRITEM;
assign DRPWDATAS7  = DRPWDATAM;

// DRP Mirrored Slave interface 8
assign DRPCLKS8    = DRPCLKM;
assign DRPRESETS8n = DRPRESETMn;
assign DRPADDRS8   = DRPADDRM;
assign DRPSELS8    = DRPSELM & (DRPSEL_Dec[8] | (DRP_Broadcast[8] & DRPWRITEM));
assign DRPWRITES8  = DRPWRITEM;
assign DRPWDATAS8  = DRPWDATAM;

// DRP Mirrored Slave interface 9
assign DRPCLKS9    = DRPCLKM;
assign DRPRESETS9n = DRPRESETMn;
assign DRPADDRS9   = DRPADDRM;
assign DRPSELS9    = DRPSELM & (DRPSEL_Dec[9] | (DRP_Broadcast[9] & DRPWRITEM));
assign DRPWRITES9  = DRPWRITEM;
assign DRPWDATAS9  = DRPWDATAM;

// DRP Mirrored Slave interface 10
assign DRPCLKS10    = DRPCLKM;
assign DRPRESETS10n = DRPRESETMn;
assign DRPADDRS10   = DRPADDRM;
assign DRPSELS10    = DRPSELM & (DRPSEL_Dec[10] | (DRP_Broadcast[10] & DRPWRITEM));
assign DRPWRITES10  = DRPWRITEM;
assign DRPWDATAS10  = DRPWDATAM;

// DRP Mirrored Slave interface 11
assign DRPCLKS11    = DRPCLKM;
```

```
assign DRPRESETS11n = DRPRESETMn;
assign DRPADDRS11   = DRPADDRM;
assign DRPSELS11    = DRPSELM & (DRPSEL_Dec[11] | (DRP_Broadcast[11] & DRPWRITEM));
assign DRPWRITES11  = DRPWRITEM;
assign DRPWDATAS11  = DRPWDATAM;

// DRP Mirrored Slave interface 12
assign DRPCLKS12    = DRPCLKM;
assign DRPRESETS12n = DRPRESETMn;
assign DRPADDRS12   = DRPADDRM;
assign DRPSELS12    = DRPSELM & (DRPSEL_Dec[12] | (DRP_Broadcast[12] & DRPWRITEM));
assign DRPWRITES12  = DRPWRITEM;
assign DRPWDATAS12  = DRPWDATAM;

// DRP Mirrored Slave interface 13
assign DRPCLKS13    = DRPCLKM;
assign DRPRESETS13n = DRPRESETMn;
assign DRPADDRS13   = DRPADDRM;
assign DRPSELS13    = DRPSELM & (DRPSEL_Dec[13] | (DRP_Broadcast[13] & DRPWRITEM));
assign DRPWRITES13  = DRPWRITEM;
assign DRPWDATAS13  = DRPWDATAM;

// DRP Mirrored Slave interface 14
assign DRPCLKS14    = DRPCLKM;
assign DRPRESETS14n = DRPRESETMn;
assign DRPADDRS14   = DRPADDRM;
assign DRPSELS14    = DRPSELM & (DRPSEL_Dec[14] | (DRP_Broadcast[14] & DRPWRITEM));
assign DRPWRITES14  = DRPWRITEM;
assign DRPWDATAS14  = DRPWDATAM;

// DRP Mirrored Slave interface 15
assign DRPCLKS15    = DRPCLKM;
assign DRPRESETS15n = DRPRESETMn;
assign DRPADDRS15   = DRPADDRM;
assign DRPSELS15    = DRPSELM & (DRPSEL_Dec[15] | (DRP_Broadcast[15] & DRPWRITEM));
assign DRPWRITES15  = DRPWRITEM;
assign DRPWDATAS15  = DRPWDATAM;

// DRP Mirrored Slave interface 16
assign DRPCLKS16    = DRPCLKM;
assign DRPRESETS16n = DRPRESETMn;
assign DRPADDRS16   = DRPADDRM;
assign DRPSELS16    = DRPSELM & (DRPSEL_Dec[16] | (DRP_Broadcast[16] & DRPWRITEM));
assign DRPWRITES16  = DRPWRITEM;
assign DRPWDATAS16  = DRPWDATAM;

// DRP Mirrored Slave interface 17
assign DRPCLKS17    = DRPCLKM;
assign DRPRESETS17n = DRPRESETMn;
assign DRPADDRS17   = DRPADDRM;
assign DRPSELS17    = DRPSELM & (DRPSEL_Dec[17] | (DRP_Broadcast[17] & DRPWRITEM));
assign DRPWRITES17  = DRPWRITEM;
assign DRPWDATAS17  = DRPWDATAM;

// DRP Mirrored Slave interface 18
assign DRPCLKS18    = DRPCLKM;
assign DRPRESETS18n = DRPRESETMn;
assign DRPADDRS18   = DRPADDRM;
assign DRPSELS18    = DRPSELM & (DRPSEL_Dec[18] | (DRP_Broadcast[18] & DRPWRITEM));
assign DRPWRITES18  = DRPWRITEM;
assign DRPWDATAS18  = DRPWDATAM;

// DRP Mirrored Slave interface 19
assign DRPCLKS19    = DRPCLKM;
assign DRPRESETS19n = DRPRESETMn;
assign DRPADDRS19   = DRPADDRM;
assign DRPSELS19    = DRPSELM & (DRPSEL_Dec[19] | (DRP_Broadcast[19] & DRPWRITEM));
assign DRPWRITES19  = DRPWRITEM;
assign DRPWDATAS19  = DRPWDATAM;
```

```
// DRP Mirrored Slave interface 20
assign DRPCLKS20    = DRPCLKM;
assign DRPRESETS20n = DRPRESETMn;
assign DRPADDRS20   = DRPADDRM;
assign DRPSELS20    = DRPSELM & (DRPSEL_Dec[20] | (DRP_Broadcast[20] & DRPWRITEM));
assign DRPWRITES20  = DRPWRITEM;
assign DRPWDATAS20  = DRPWDATAM;

// DRP Mirrored Slave interface 21
assign DRPCLKS21    = DRPCLKM;
assign DRPRESETS21n = DRPRESETMn;
assign DRPADDRS21   = DRPADDRM;
assign DRPSELS21    = DRPSELM & (DRPSEL_Dec[21] | (DRP_Broadcast[21] & DRPWRITEM));
assign DRPWRITES21  = DRPWRITEM;
assign DRPWDATAS21  = DRPWDATAM;

// DRP Mirrored Slave interface 22
assign DRPCLKS22    = DRPCLKM;
assign DRPRESETS22n = DRPRESETMn;
assign DRPADDRS22   = DRPADDRM;
assign DRPSELS22    = DRPSELM & (DRPSEL_Dec[22] | (DRP_Broadcast[22] & DRPWRITEM));
assign DRPWRITES22  = DRPWRITEM;
assign DRPWDATAS22  = DRPWDATAM;

// DRP Mirrored Slave interface 23
assign DRPCLKS23    = DRPCLKM;
assign DRPRESETS23n = DRPRESETMn;
assign DRPADDRS23   = DRPADDRM;
assign DRPSELS23    = DRPSELM & (DRPSEL_Dec[23] | (DRP_Broadcast[23] & DRPWRITEM));
assign DRPWRITES23  = DRPWRITEM;
assign DRPWDATAS23  = DRPWDATAM;

// DRP Mirrored Slave interface 24
assign DRPCLKS24    = DRPCLKM;
assign DRPRESETS24n = DRPRESETMn;
assign DRPADDRS24   = DRPADDRM;
assign DRPSELS24    = DRPSELM & (DRPSEL_Dec[24] | (DRP_Broadcast[24] & DRPWRITEM));
assign DRPWRITES24  = DRPWRITEM;
assign DRPWDATAS24  = DRPWDATAM;

// DRP Mirrored Slave interface 25
assign DRPCLKS25    = DRPCLKM;
assign DRPRESETS25n = DRPRESETMn;
assign DRPADDRS25   = DRPADDRM;
assign DRPSELS25    = DRPSELM & (DRPSEL_Dec[25] | (DRP_Broadcast[25] & DRPWRITEM));
assign DRPWRITES25  = DRPWRITEM;
assign DRPWDATAS25  = DRPWDATAM;

// DRP Mirrored Slave interface 26
assign DRPCLKS26    = DRPCLKM;
assign DRPRESETS26n = DRPRESETMn;
assign DRPADDRS26   = DRPADDRM;
assign DRPSELS26    = DRPSELM & (DRPSEL_Dec[26] | (DRP_Broadcast[26] & DRPWRITEM));
assign DRPWRITES26  = DRPWRITEM;
assign DRPWDATAS26  = DRPWDATAM;

// DRP Mirrored Slave interface 27
assign DRPCLKS27    = DRPCLKM;
assign DRPRESETS27n = DRPRESETMn;
assign DRPADDRS27   = DRPADDRM;
assign DRPSELS27    = DRPSELM & (DRPSEL_Dec[27] | (DRP_Broadcast[27] & DRPWRITEM));
assign DRPWRITES27  = DRPWRITEM;
assign DRPWDATAS27  = DRPWDATAM;

// DRP Mirrored Slave interface 28
assign DRPCLKS28    = DRPCLKM;
assign DRPRESETS28n = DRPRESETMn;
assign DRPADDRS28   = DRPADDRM;
assign DRPSELS28    = DRPSELM & (DRPSEL_Dec[28] | (DRP_Broadcast[28] & DRPWRITEM));
assign DRPWRITES28  = DRPWRITEM;
```

```
    assign DRPWDATAS28  = DRPWDATAM;

    // DRP Mirrored Slave interface 29
    assign DRPCLKS29    = DRPCLKM;
    assign DRPRESETS29n = DRPRESETMn;
    assign DRPADDRS29   = DRPADDRM;
    assign DRPSELS29    = DRPSELM & (DRPSEL_Dec[29] | (DRP_Broadcast[29] & DRPWRITEM));
    assign DRPWRITES29  = DRPWRITEM;
    assign DRPWDATAS29  = DRPWDATAM;

    // DRP Mirrored Slave interface 30
    assign DRPCLKS30    = DRPCLKM;
    assign DRPRESETS30n = DRPRESETMn;
    assign DRPADDRS30   = DRPADDRM;
    assign DRPSELS30    = DRPSELM & (DRPSEL_Dec[30] | (DRP_Broadcast[30] & DRPWRITEM));
    assign DRPWRITES30  = DRPWRITEM;
    assign DRPWDATAS30  = DRPWDATAM;

    // DRP Mirrored Slave interface 31
    assign DRPCLKS31    = DRPCLKM;
    assign DRPRESETS31n = DRPRESETMn;
    assign DRPADDRS31   = DRPADDRM;
    assign DRPSELS31    = DRPSELM & (DRPSEL_Dec[31] | (DRP_Broadcast[31] & DRPWRITEM));
    assign DRPWRITES31  = DRPWRITEM;
    assign DRPWDATAS31  = DRPWDATAM;



module drpChannelPort (
    DRPCLK,
    DRPRESETn,
    DRPADDR,
    DRPSEL,
    DRPWRITE,
    DRPWDATA,
    DRPRDATA,
    DRPREADY,

    X_DRPCLK,
    X_DRPRESETn,
    X_DRPADDR,
    X_DRPSEL,
    X_DRPWRITE,
    X_DRPWDATA,
    X_DRPRDATA,
    X_DRPREADY,

    asRstN,

    asBroadcast,
    asGpoMuxSel,
    asSoftRst,
    asLoopback,
    asTxSysClkSel,
    asRxSysClkSel,
    asTxOutClkSel,
    asRxOutClkSel,
    asTxDiffCtrl,
    asTxMainCursor,
    asTxPreCursor,
    asTxPostCursor,

    txUsrClk2,
    txUsrRstN,
    txRdy,

    txData,
    txPrbsSel,
    txpd,
    txPrbsForceErrLvl,
```

```
    txPrbsForceErrPls,
    txPolarity,
    tx8b10bEn,
    txRate,

    txResetDone,

    rxUsrClk2,
    rxUsrRstN,
    rxRdy,

    rxData,
    rxPrbsSel,
    rxPrbsCntRstPls,
    rxPolarity,
    rx8b10bEn,
    rxRate,

    rxPrbsErrFlag,
    rxResetDone
);

input  DRPCLK;
input  DRPRESETn;
input  [31:0] DRPADDR;
input  DRPSEL;
input  DRPWRITE;
input  [63:0] DRPWDATA;
output [63:0] DRPRDATA;
output DRPREADY;

output X_DRPCLK;
output X_DRPRESETn;
output [31:0] X_DRPADDR;
output X_DRPSEL;
output X_DRPWRITE;
output [63:0] X_DRPWDATA;
input  [63:0] X_DRPRDATA;
input  X_DRPREADY;

input asRstN;

output [31:0] asBroadcast;
output  [3:0] asGpoMuxSel;
output        asSoftRst;
output  [2:0] asLoopback;
output  [1:0] asTxSysClkSel;
output  [1:0] asRxSysClkSel;
output  [2:0] asTxOutClkSel;
output  [2:0] asRxOutClkSel;
output  [3:0] asTxDiffCtrl;
output  [6:0] asTxMainCursor;
output  [4:0] asTxPreCursor;
output  [4:0] asTxPostCursor;

input  txUsrClk2;
input  txUsrRstN;
input  txRdy;

output [63:0] txData;
output  [2:0] txPrbsSel;
output  [1:0] txpd;
output txPrbsForceErrLvl;
output txPrbsForceErrPls;
output txPolarity;
output tx8b10bEn;
output  [2:0] txRate;

input  txResetDone;
```

```
    input  rxUsrClk2;
    input  rxUsrRstN;
    input  rxRdy;

    input  [63:0] rxData;
    output  [2:0] rxPrbsSel;
    output rxPrbsCntRstPls;
    output rxPolarity;
    output rx8b10bEn;
    output  [2:0] rxRate;

    input  rxPrbsErrFlag;
    input  rxResetDone;

    parameter INIT     = 2'd0;
    parameter WAIT_END = 2'd1;
    parameter DONE     = 2'd2;

    reg [1:0] state;
    reg drpActive;
    reg [31:0] DRPADDRQ;
    reg DRPWRITEQ;
    reg [63:0] DRPWDATAQ;
    reg DRPSELQ;
    reg txRdy_syncQ;
    reg rxRdy_syncQ;

    wire txRdy_sync;
    wire rxRdy_sync;

    wire [63:0] DRPRDATA_reg;
    wire [63:0] DRPRDATA_drp;
    wire [63:0] DRPRDATA_tx;
    wire [63:0] DRPRDATA_rx;

    reg [3:0] subChannelPortDecode;

    wire DRPREADY_reg;
    wire DRPREADY_drp;
    wire DRPREADY_tx;
    wire DRPREADY_rx;

    reg [63:0] DRPRDATA_int;
    reg        DRPREADY_int;

    reg [63:0] DRPRDATA;
    reg        DRPREADY;

    assign DRPREADY_tx  = txRdy_sync & ~txRdy_syncQ;
    assign DRPREADY_rx  = rxRdy_sync & ~rxRdy_syncQ;

    assign X_DRPCLK     = DRPCLK;
    assign X_DRPRESETn  = DRPRESETn;
    assign X_DRPADDR    = DRPADDRQ;
    assign X_DRPWRITE   = DRPSELQ & DRPWRITEQ & subChannelPortDecode[1];
    assign X_DRPSEL     = DRPSELQ & subChannelPortDecode[1];
    assign X_DRPWDATA   = DRPWDATAQ;
    assign DRPRDATA_drp = X_DRPRDATA;
    assign DRPREADY_drp = X_DRPREADY;

    wire [1:0] decodeAddrBits = DRPADDRQ[23:22];
    always @ (drpActive or decodeAddrBits or
              DRPRDATA_reg or DRPRDATA_drp or DRPRDATA_tx or DRPRDATA_rx or
              DRPREADY_reg or DRPREADY_drp or DRPREADY_tx or DRPREADY_rx) begin
       subChannelPortDecode = 4'b0000;
       if (drpActive) begin
          case (decodeAddrBits)
          2'b00: begin subChannelPortDecode[0] = 1'b1; DRPRDATA_int = DRPRDATA_reg;  DRPREADY_int = DRPREADY_reg;
end //local registers
```

```
        2'b01: begin subChannelPortDecode[1] = 1'b1; DRPRDATA_int = DRPRDATA_drp;  DRPREADY_int = DRPREADY_drp;
end //Xilinx DRP registers
        2'b10: begin subChannelPortDecode[2] = 1'b1; DRPRDATA_int = DRPRDATA_tx;    DRPREADY_int = DRPREADY_tx;
end //asynchronous tx interface
        2'b11: begin subChannelPortDecode[3] = 1'b1; DRPRDATA_int = DRPRDATA_rx;    DRPREADY_int = DRPREADY_rx;
end //asynchronous rx interface
          endcase
       end
       else begin
          subChannelPortDecode = 4'b0000;
          DRPREADY_int  = 1'b0;
          DRPRDATA_int  = 64'd0;
       end
    end

    always @ (posedge DRPCLK or negedge DRPRESETn) begin
       if (~DRPRESETn) begin
          state     <= INIT;
          drpActive <= 1'b0;
          DRPSELQ   <= 1'b0;
          DRPADDRQ  <= 32'd0;
          DRPWRITEQ <= 1'b0;
          DRPWDATAQ <= 64'd0;
          DRPRDATA  <= 64'd0;

          txRdy_syncQ <= 1'b0;
          rxRdy_syncQ <= 1'b0;
       end
       else begin

          txRdy_syncQ <= txRdy_sync;
          rxRdy_syncQ <= rxRdy_sync;

          case (state)
          INIT: begin
             DRPREADY  <= 1'b0;
             if (DRPSEL) begin
                drpActive <= 1'b1;

                DRPSELQ   <= 1'b1;
                DRPWRITEQ <= DRPWRITE;
                DRPADDRQ  <= DRPADDR;
                DRPWDATAQ <= DRPWDATA;

                state     <= WAIT_END;
             end
             else begin
                drpActive <= 1'b0;

                DRPSELQ   <= 1'b0;
                DRPWRITEQ <= 1'b0;
                DRPADDRQ  <= 0;
                DRPWDATAQ <= 0;
             end
          end
          WAIT_END: begin
             DRPSELQ       <= 1'b0;
             if (DRPREADY_int) begin
                drpActive <= 1'b0;

                DRPREADY  <= 1'b1;
                DRPRDATA  <= DRPRDATA_int;

                state     <= DONE;
             end
             else begin
                DRPREADY  <= 1'b0;
             end
          end
          DONE: begin
```

```
            DRPREADY  <= 1'b0;
            DRPSELQ   <= 1'b0;
            DRPWRITEQ <= 1'b0;
            DRPADDRQ  <= 0;
            DRPWDATAQ <= 0;

            if (!DRPSEL) begin
               state     <= INIT;
            end
         end
         default: begin
            state <= INIT;
         end
         endcase
      end
   end

   drpRegIf drpRegIf (
      .drpClk           (DRPCLK),
      .drpResetN        (asRstN),
      .drpSel           (DRPSELQ & subChannelPortDecode[0]),
      .drpWrite         (DRPSELQ & subChannelPortDecode[0] & DRPWRITEQ),
      .drpAddr          (DRPADDRQ),
      .drpWdata         (DRPWDATAQ),
      .drpRdata         (DRPRDATA_reg),
      .drpReady         (DRPREADY_reg),

      .regBroadcast     (asBroadcast),
      .regGpoMuxSel     (asGpoMuxSel),
      .regSoftRst       (asSoftRst),
      .regLoopback      (asLoopback),
      .regTxSysClkSel   (asTxSysClkSel),
      .regRxSysClkSel   (asRxSysClkSel),
      .regTxOutClkSel   (asTxOutClkSel),
      .regRxOutClkSel   (asRxOutClkSel),
      .regTxDiffCtrl    (asTxDiffCtrl),
      .regTxMainCursor  (asTxMainCursor),
      .regTxPreCursor   (asTxPreCursor),
      .regTxPostCursor  (asTxPostCursor)
   );

   drpTxRegIf drpTxRegIf (
      .usrClk           (txUsrClk2),
      .usrRstN          (txUsrRstN),
      .enable           (txRdy),

      .drpSel_async     (drpActive & subChannelPortDecode[2]),
      .drpWrite         (DRPWRITEQ),
      .drpAddr          (DRPADDRQ),
      .drpWdata         (DRPWDATAQ),
      .drpRdata         (DRPRDATA_tx),
      .drpReady         (txRdy_async),

      .txData           (txData),
      .txPrbsSel        (txPrbsSel),
      .txpd             (txpd),
      .txPrbsForceErrLvl (txPrbsForceErrLvl),
      .txPrbsForceErrPls (txPrbsForceErrPls),
      .txPolarity       (txPolarity),
      .tx8b10bEn        (tx8b10bEn),
      .txRate           (txRate),

      .txResetDone      (txResetDone)
   );

   drpRxRegIf drpRxRegIf (
      .usrClk           (rxUsrClk2),
      .usrRstN          (rxUsrRstN),
      .enable           (rxRdy),
```

```
    .drpSel_async      (drpActive & subChannelPortDecode[3]),
    .drpWrite          (DRPWRITEQ),
    .drpAddr           (DRPADDRQ),
    .drpWdata          (DRPWDATAQ),
    .drpRdata          (DRPRDATA_rx),
    .drpReady          (rxRdy_async),

    .rxData            (rxData),
    .rxPrbsSel         (rxPrbsSel),
    .rxPrbsCntRstPls   (rxPrbsCntRstPls),
    .rxPolarity        (rxPolarity),
    .rx8b10bEn         (rx8b10bEn),
    .rxRate            (rxRate),

    .rxPrbsErrFlag     (rxPrbsErrFlag),
    .rxResetDone       (rxResetDone)
);

socSync socSync_txRdy (
    .outClk            (DRPCLK),
    .outRstN           (DRPRESETn),
    .asyncInput        (txRdy_async),
    .syncOutput        (txRdy_sync)
);

socSync socSync_rxRdy (
    .outClk            (DRPCLK),
    .outRstN           (DRPRESETn),
    .asyncInput        (rxRdy_async),
    .syncOutput        (rxRdy_sync)
);

Endmodule


module gtx8_chan_pll_init #
(
    parameter EXAMPLE_SIM_GTRESET_SPEEDUP          = "TRUE",              // Simulation setting for GT SecureIP model
    parameter EXAMPLE_SIMULATION                   =  0,                  // Set to 1 for simulation
    parameter STABLE_CLOCK_PERIOD                  =  20, // Period of the stable clock driving this
                                                          // state-machine, unit is [ns]

    parameter EXAMPLE_USE_CHIPSCOPE                =  0                  // Set to 1 to use Chipscope to drive resets

)
(
    input          SYSCLK_IN,
    input          SOFT_RESET_IN,
    input          DONT_RESET_ON_DATA_ERROR_IN,
    output         GT0_TX_FSM_RESET_DONE_OUT,
    output         GT0_RX_FSM_RESET_DONE_OUT,
    input          GT0_DATA_VALID_IN,
    output         GT1_TX_FSM_RESET_DONE_OUT,
    output         GT1_RX_FSM_RESET_DONE_OUT,
    input          GT1_DATA_VALID_IN,
    output         GT2_TX_FSM_RESET_DONE_OUT,
    output         GT2_RX_FSM_RESET_DONE_OUT,
    input          GT2_DATA_VALID_IN,
    output         GT3_TX_FSM_RESET_DONE_OUT,
    output         GT3_RX_FSM_RESET_DONE_OUT,
    input          GT3_DATA_VALID_IN,

    //_____________________________________________________________________
    //GT0  (X1Y12)
    //____________________________CHANNEL PORTS____________________________
    //--------------------------- Channel - DRP Ports  --------------------------
    input   [8:0]   GT0_DRPADDR_IN,
    input           GT0_DRPCLK_IN,
    input   [15:0]  GT0_DRPDI_IN,
    output  [15:0]  GT0_DRPDO_OUT,
```

```
input          GT0_DRPEN_IN,
output         GT0_DRPRDY_OUT,
input          GT0_DRPWE_IN,

input   [2:0]   GT0_RXOUTCLKSEL_IN,
input   [2:0]   GT0_TXOUTCLKSEL_IN,
input   [1:0]   GT0_RXSYSCLKSEL_IN,
input   [1:0]   GT0_TXSYSCLKSEL_IN,
input          GT0_TX8B10BEN_IN,
input          GT0_RX8B10BEN_IN,
output         GT0_RXOUTCLK_OUT,
//------------------------ Channel - Clocking Ports ------------------------
input          gt0_gtrefclk0_in,
//------------------------------ Loopback Ports ------------------------------
input   [2:0]   GT0_LOOPBACK_IN,
//---------------------------- PCI Express Ports ----------------------------
input   [2:0]   GT0_RXRATE_IN,
//-------------------- RX Initialization and Reset Ports --------------------
input          GT0_RXUSERRDY_IN,
//------------------------ RX Margin Analysis Ports ------------------------
output         GT0_EYESCANDATAERROR_OUT,
//------------------------ Receive Ports - CDR Ports ------------------------
output         GT0_RXCDRLOCK_OUT,
//----------------- Receive Ports - FPGA RX Interface Ports -----------------
input          GT0_RXUSRCLK_IN,
input          GT0_RXUSRCLK2_IN,
//----------------- Receive Ports - FPGA RX interface Ports -----------------
output  [31:0]  GT0_RXDATA_OUT,
//----------------- Receive Ports - Pattern Checker Ports ------------------
output         GT0_RXPRBSERR_OUT,
input   [2:0]   GT0_RXPRBSSEL_IN,
//------------------ Receive Ports - Pattern Checker ports ------------------
input          GT0_RXPRBSCNTRESET_IN,
//------------------------ Receive Ports - RX AFE -------------------------
input          GT0_GTXRXP_IN,
//---------------------- Receive Ports - RX AFE Ports ----------------------
input          GT0_GTXRXN_IN,
//---------- Receive Ports - RX Fabric ClocK Output Control Ports ----------
output         GT0_RXRATEDONE_OUT,
//----------- Receive Ports - RX Initialization and Reset Ports ------------
input          GT0_GTRXRESET_IN,
input          GT0_RXPMARESET_IN,
//---------------- Receive Ports - RX Polarity Control Ports ----------------
input          GT0_RXPOLARITY_IN,
//-------------------- Receive Ports - RX gearbox ports --------------------
input          GT0_RXSLIDE_IN,
//------------ Receive Ports -RX Initialization and Reset Ports ------------
output         GT0_RXRESETDONE_OUT,
//---------------------- TX Configurable Driver Ports ----------------------
input   [4:0]   GT0_TXPOSTCURSOR_IN,
input   [4:0]   GT0_TXPRECURSOR_IN,
//------------------- TX Initialization and Reset Ports --------------------
input          GT0_GTTXRESET_IN,
input          GT0_TXUSERRDY_IN,
//----------------- Transmit Ports - FPGA TX Interface Ports -----------------
input          GT0_TXUSRCLK_IN,
input          GT0_TXUSRCLK2_IN,
//------------------ Transmit Ports - PCI Express Ports -------------------
input   [2:0]   GT0_TXRATE_IN,
//----------------- Transmit Ports - Pattern Generator Ports -----------------
input          GT0_TXPRBSFORCEERR_IN,
//-------------- Transmit Ports - TX Configurable Driver Ports --------------
input   [3:0]   GT0_TXDIFFCTRL_IN,
input   [6:0]   GT0_TXMAINCURSOR_IN,
//----------------- Transmit Ports - TX Data Path interface -----------------
input   [31:0]  GT0_TXDATA_IN,
//--------------- Transmit Ports - TX Driver and OOB signaling ---------------
output         GT0_GTXTXN_OUT,
output         GT0_GTXTXP_OUT,
//---------- Transmit Ports - TX Fabric Clock Output Control Ports ----------
```

```
output         GT0_TXOUTCLK_OUT,
output         GT0_TXOUTCLKFABRIC_OUT,
output         GT0_TXOUTCLKPCS_OUT,
output         GT0_TXRATEDONE_OUT,
//----------- Transmit Ports - TX Initialization and Reset Ports -----------
output         GT0_TXRESETDONE_OUT,
//---------------- Transmit Ports - TX Polarity Control Ports ----------------
input          GT0_TXPOLARITY_IN,
//---------------- Transmit Ports - pattern Generator Ports ----------------
input   [2:0]   GT0_TXPRBSSEL_IN,
input   [1:0]   GT0_TXPD_IN,

//GT1  (X1Y13)
//____________________________________CHANNEL PORTS____________________________________
//--------------------------- Channel - DRP Ports  ---------------------------
input   [8:0]   GT1_DRPADDR_IN,
input          GT1_DRPCLK_IN,
input   [15:0]  GT1_DRPDI_IN,
output  [15:0]  GT1_DRPDO_OUT,
input          GT1_DRPEN_IN,
output         GT1_DRPRDY_OUT,
input          GT1_DRPWE_IN,

input   [2:0]   GT1_RXOUTCLKSEL_IN,
input   [2:0]   GT1_TXOUTCLKSEL_IN,
input   [1:0]   GT1_RXSYSCLKSEL_IN,
input   [1:0]   GT1_TXSYSCLKSEL_IN,
input          GT1_TX8B10BEN_IN,
input          GT1_RX8B10BEN_IN,
output         GT1_RXOUTCLK_OUT,
//------------------------- Channel - Clocking Ports -------------------------
input          gt1_gtrefclk0_in,
//---------------------------- Loopback Ports -----------------------------
input   [2:0]   GT1_LOOPBACK_IN,
//---------------------------- PCI Express Ports ----------------------------
input   [2:0]   GT1_RXRATE_IN,
//------------------- RX Initialization and Reset Ports --------------------
input          GT1_RXUSERRDY_IN,
//------------------------ RX Margin Analysis Ports ------------------------
output         GT1_EYESCANDATAERROR_OUT,
//----------------------- Receive Ports - CDR Ports ------------------------
output         GT1_RXCDRLOCK_OUT,
//---------------- Receive Ports - FPGA RX Interface Ports -----------------
input          GT1_RXUSRCLK_IN,
input          GT1_RXUSRCLK2_IN,
//---------------- Receive Ports - FPGA RX interface Ports -----------------
output  [31:0]  GT1_RXDATA_OUT,
//----------------- Receive Ports - Pattern Checker Ports ------------------
output         GT1_RXPRBSERR_OUT,
input   [2:0]   GT1_RXPRBSSEL_IN,
//----------------- Receive Ports - Pattern Checker ports ------------------
input          GT1_RXPRBSCNTRESET_IN,
//------------------------ Receive Ports - RX AFE ------------------------
input          GT1_GTXRXP_IN,
//---------------------- Receive Ports - RX AFE Ports ----------------------
input          GT1_GTXRXN_IN,
//---------- Receive Ports - RX Fabric ClocK Output Control Ports ----------
output         GT1_RXRATEDONE_OUT,
//----------- Receive Ports - RX Initialization and Reset Ports ------------
input          GT1_GTRXRESET_IN,
input          GT1_RXPMARESET_IN,
//---------------- Receive Ports - RX Polarity Control Ports ----------------
input          GT1_RXPOLARITY_IN,
//--------------------- Receive Ports - RX gearbox ports ---------------------
input          GT1_RXSLIDE_IN,
//------------ Receive Ports -RX Initialization and Reset Ports ------------
output         GT1_RXRESETDONE_OUT,
//---------------------- TX Configurable Driver Ports ----------------------
input   [4:0]   GT1_TXPOSTCURSOR_IN,
input   [4:0]   GT1_TXPRECURSOR_IN,
```

```
//------------------- TX Initialization and Reset Ports --------------------
input          GT1_GTTXRESET_IN,
input          GT1_TXUSERRDY_IN,
//---------------- Transmit Ports - FPGA TX Interface Ports ----------------
input          GT1_TXUSRCLK_IN,
input          GT1_TXUSRCLK2_IN,
//------------------- Transmit Ports - PCI Express Ports -------------------
input   [2:0]   GT1_TXRATE_IN,
//---------------- Transmit Ports - Pattern Generator Ports ----------------
input          GT1_TXPRBSFORCEERR_IN,
//------------- Transmit Ports - TX Configurable Driver Ports --------------
input   [3:0]   GT1_TXDIFFCTRL_IN,
input   [6:0]   GT1_TXMAINCURSOR_IN,
//---------------- Transmit Ports - TX Data Path interface -----------------
input   [31:0]  GT1_TXDATA_IN,
//-------------- Transmit Ports - TX Driver and OOB signaling --------------
output          GT1_GTXTXN_OUT,
output          GT1_GTXTXP_OUT,
//--------- Transmit Ports - TX Fabric Clock Output Control Ports ----------
output          GT1_TXOUTCLK_OUT,
output          GT1_TXOUTCLKFABRIC_OUT,
output          GT1_TXOUTCLKPCS_OUT,
output          GT1_TXRATEDONE_OUT,
//----------- Transmit Ports - TX Initialization and Reset Ports -----------
output          GT1_TXRESETDONE_OUT,
//---------------- Transmit Ports - TX Polarity Control Ports ----------------
input          GT1_TXPOLARITY_IN,
//---------------- Transmit Ports - pattern Generator Ports ----------------
input   [2:0]   GT1_TXPRBSSEL_IN,
input   [1:0]   GT1_TXPD_IN,

//GT2  (X1Y14)
//_____________________________CHANNEL PORTS_____________________________
//-------------------------- Channel - DRP Ports  --------------------------
input   [8:0]   GT2_DRPADDR_IN,
input          GT2_DRPCLK_IN,
input   [15:0]  GT2_DRPDI_IN,
output  [15:0]  GT2_DRPDO_OUT,
input          GT2_DRPEN_IN,
output          GT2_DRPRDY_OUT,
input          GT2_DRPWE_IN,

input   [2:0]   GT2_RXOUTCLKSEL_IN,
input   [2:0]   GT2_TXOUTCLKSEL_IN,
input   [1:0]   GT2_RXSYSCLKSEL_IN,
input   [1:0]   GT2_TXSYSCLKSEL_IN,
input          GT2_TX8B10BEN_IN,
input          GT2_RX8B10BEN_IN,
output          GT2_RXOUTCLK_OUT,
//------------------------ Channel - Clocking Ports ------------------------
input          gt2_gtrefclk0_in,
//-------------------------- Loopback Ports ----------------------------
input   [2:0]   GT2_LOOPBACK_IN,
//--------------------------- PCI Express Ports ----------------------------
input   [2:0]   GT2_RXRATE_IN,
//------------------- RX Initialization and Reset Ports --------------------
input          GT2_RXUSERRDY_IN,
//------------------------ RX Margin Analysis Ports ------------------------
output          GT2_EYESCANDATAERROR_OUT,
//----------------------- Receive Ports - CDR Ports ------------------------
output          GT2_RXCDRLOCK_OUT,
//----------------- Receive Ports - FPGA RX Interface Ports -----------------
input          GT2_RXUSRCLK_IN,
input          GT2_RXUSRCLK2_IN,
//---------------- Receive Ports - FPGA RX interface Ports -----------------
output  [31:0]  GT2_RXDATA_OUT,
//----------------- Receive Ports - Pattern Checker Ports ------------------
output          GT2_RXPRBSERR_OUT,
input   [2:0]   GT2_RXPRBSSEL_IN,
//----------------- Receive Ports - Pattern Checker ports ------------------
```

```
input          GT2_RXPRBSCNTRESET_IN,
//------------------------- Receive Ports - RX AFE -------------------------
input          GT2_GTXRXP_IN,
//---------------------- Receive Ports - RX AFE Ports ----------------------
input          GT2_GTXRXN_IN,
//---------- Receive Ports - RX Fabric ClocK Output Control Ports ----------
output         GT2_RXRATEDONE_OUT,
//----------- Receive Ports - RX Initialization and Reset Ports ------------
input          GT2_GTRXRESET_IN,
input          GT2_RXPMARESET_IN,
//--------------- Receive Ports - RX Polarity Control Ports ----------------
input          GT2_RXPOLARITY_IN,
//--------------------- Receive Ports - RX gearbox ports --------------------
input          GT2_RXSLIDE_IN,
//------------ Receive Ports -RX Initialization and Reset Ports ------------
output         GT2_RXRESETDONE_OUT,
//---------------------- TX Configurable Driver Ports ----------------------
input   [4:0]   GT2_TXPOSTCURSOR_IN,
input   [4:0]   GT2_TXPRECURSOR_IN,
//------------------- TX Initialization and Reset Ports --------------------
input          GT2_GTTXRESET_IN,
input          GT2_TXUSERRDY_IN,
//---------------- Transmit Ports - FPGA TX Interface Ports ----------------
input          GT2_TXUSRCLK_IN,
input          GT2_TXUSRCLK2_IN,
//------------------- Transmit Ports - PCI Express Ports -------------------
input   [2:0]   GT2_TXRATE_IN,
//---------------- Transmit Ports - Pattern Generator Ports ----------------
input          GT2_TXPRBSFORCEERR_IN,
//------------- Transmit Ports - TX Configurable Driver Ports --------------
input   [3:0]   GT2_TXDIFFCTRL_IN,
input   [6:0]   GT2_TXMAINCURSOR_IN,
//---------------- Transmit Ports - TX Data Path interface -----------------
input   [31:0]  GT2_TXDATA_IN,
//-------------- Transmit Ports - TX Driver and OOB signaling --------------
output         GT2_GTXTXN_OUT,
output         GT2_GTXTXP_OUT,
//--------- Transmit Ports - TX Fabric Clock Output Control Ports ----------
output         GT2_TXOUTCLK_OUT,
output         GT2_TXOUTCLKFABRIC_OUT,
output         GT2_TXOUTCLKPCS_OUT,
output         GT2_TXRATEDONE_OUT,
//----------- Transmit Ports - TX Initialization and Reset Ports -----------
output         GT2_TXRESETDONE_OUT,
//--------------- Transmit Ports - TX Polarity Control Ports ---------------
input          GT2_TXPOLARITY_IN,
//---------------- Transmit Ports - pattern Generator Ports ----------------
input   [2:0]   GT2_TXPRBSSEL_IN,
input   [1:0]   GT2_TXPD_IN,

//GT3  (X1Y15)
//_____________________________CHANNEL PORTS_____________________________
//------------------------- Channel - DRP Ports  -------------------------
input   [8:0]   GT3_DRPADDR_IN,
input          GT3_DRPCLK_IN,
input   [15:0]  GT3_DRPDI_IN,
output  [15:0]  GT3_DRPDO_OUT,
input          GT3_DRPEN_IN,
output         GT3_DRPRDY_OUT,
input          GT3_DRPWE_IN,

input   [2:0]   GT3_RXOUTCLKSEL_IN,
input   [2:0]   GT3_TXOUTCLKSEL_IN,
input   [1:0]   GT3_RXSYSCLKSEL_IN,
input   [1:0]   GT3_TXSYSCLKSEL_IN,
input          GT3_TX8B10BEN_IN,
input          GT3_RX8B10BEN_IN,
output         GT3_RXOUTCLK_OUT,
//----------------------- Channel - Clocking Ports -----------------------
input          gt3_gtrefclk0_in,
```

```
//------------------------------ Loopback Ports -----------------------------
input   [2:0]   GT3_LOOPBACK_IN,
//------------------------------ PCI Express Ports -----------------------------
input   [2:0]   GT3_RXRATE_IN,
//------------------- RX Initialization and Reset Ports --------------------
input           GT3_RXUSERRDY_IN,
//------------------------ RX Margin Analysis Ports ------------------------
output          GT3_EYESCANDATAERROR_OUT,
//----------------------- Receive Ports - CDR Ports ------------------------
output          GT3_RXCDRLOCK_OUT,
//----------------- Receive Ports - FPGA RX Interface Ports ------------------
input           GT3_RXUSRCLK_IN,
input           GT3_RXUSRCLK2_IN,
//----------------- Receive Ports - FPGA RX interface Ports ------------------
output  [31:0]  GT3_RXDATA_OUT,
//------------------ Receive Ports - Pattern Checker Ports -------------------
output          GT3_RXPRBSERR_OUT,
input   [2:0]   GT3_RXPRBSSEL_IN,
//------------------ Receive Ports - Pattern Checker ports -------------------
input           GT3_RXPRBSCNTRESET_IN,
//------------------------ Receive Ports - RX AFE -------------------------
input           GT3_GTXRXP_IN,
//----------------------- Receive Ports - RX AFE Ports -----------------------
input           GT3_GTXRXN_IN,
//---------- Receive Ports - RX Fabric ClocK Output Control Ports ----------
output          GT3_RXRATEDONE_OUT,
//----------- Receive Ports - RX Initialization and Reset Ports ------------
input           GT3_GTRXRESET_IN,
input           GT3_RXPMARESET_IN,
//---------------- Receive Ports - RX Polarity Control Ports -----------------
input           GT3_RXPOLARITY_IN,
//--------------------- Receive Ports - RX gearbox ports ---------------------
input           GT3_RXSLIDE_IN,
//------------ Receive Ports -RX Initialization and Reset Ports ------------
output          GT3_RXRESETDONE_OUT,
//----------------------- TX Configurable Driver Ports -----------------------
input   [4:0]   GT3_TXPOSTCURSOR_IN,
input   [4:0]   GT3_TXPRECURSOR_IN,
//------------------- TX Initialization and Reset Ports --------------------
input           GT3_GTTXRESET_IN,
input           GT3_TXUSERRDY_IN,
//----------------- Transmit Ports - FPGA TX Interface Ports -----------------
input           GT3_TXUSRCLK_IN,
input           GT3_TXUSRCLK2_IN,
//------------------- Transmit Ports - PCI Express Ports -------------------
input   [2:0]   GT3_TXRATE_IN,
//----------------- Transmit Ports - Pattern Generator Ports -----------------
input           GT3_TXPRBSFORCEERR_IN,
//------------- Transmit Ports - TX Configurable Driver Ports --------------
input   [3:0]   GT3_TXDIFFCTRL_IN,
input   [6:0]   GT3_TXMAINCURSOR_IN,
//----------------- Transmit Ports - TX Data Path interface -----------------
input   [31:0]  GT3_TXDATA_IN,
//--------------- Transmit Ports - TX Driver and OOB signaling ---------------
output          GT3_GTXTXN_OUT,
output          GT3_GTXTXP_OUT,
//--------- Transmit Ports - TX Fabric Clock Output Control Ports ----------
output          GT3_TXOUTCLK_OUT,
output          GT3_TXOUTCLKFABRIC_OUT,
output          GT3_TXOUTCLKPCS_OUT,
output          GT3_TXRATEDONE_OUT,
//------------ Transmit Ports - TX Initialization and Reset Ports ------------
output          GT3_TXRESETDONE_OUT,
//----------------- Transmit Ports - TX Polarity Control Ports -----------------
input           GT3_TXPOLARITY_IN,
//----------------- Transmit Ports - pattern Generator Ports -----------------
input   [2:0]   GT3_TXPRBSSEL_IN,
input   [1:0]   GT3_TXPD_IN,
```

```
    //_____________________________COMMON PORTS_____________________________
    //--------------------- Common Block  - Ref Clock Ports ---------------------
    input          GT0_GTREFCLK0_COMMON_IN,
    //----------------------- Common Block - QPLL Ports -------------------------
    output          GT0_QPLLLOCK_OUT,
    input          GT0_QPLLLOCKDETCLK_IN,
    input          GT0_QPLLRESET_IN,

    input   [7:0]   GTC0_DRPADDR,
    input          GTC0_DRPCLK,
    input  [15:0]   GTC0_DRPDI,
    output [15:0]   GTC0_DRPDO,
    input          GTC0_DRPEN,
    output          GTC0_DRPRDY,
    input          GTC0_DRPWE


);



//*************************************Parameter Declarations**********************

    //Typical CDRLOCK Time is 50,000UI, as per DS183
    parameter RX_CDRLOCK_TIME      = (EXAMPLE_SIMULATION == 1) ? 1000 : 50000/6.4;


    integer   WAIT_TIME_CDRLOCK    = RX_CDRLOCK_TIME / STABLE_CLOCK_PERIOD;

//--------------------------- GT Wrapper Wires ------------------------------
    wire          gt0_txresetdone_i;
    wire          gt0_rxresetdone_i;
    wire          gt0_gttxreset_i;
    wire          gt0_gttxreset_t;
    wire          gt0_gtrxreset_i;
    wire          gt0_gtrxreset_t;
    wire          gt0_rxdfelpmreset_i;
    wire   [1:0]   gt0_txsysclksel_i;
    wire   [1:0]   gt0_rxsysclksel_i;
    wire          gt0_txuserrdy_i;
    wire          gt0_txuserrdy_t;
    wire          gt0_rxuserrdy_i;
    wire          gt0_rxuserrdy_t;

    wire          gt0_rxdfeagchold_i;
    wire          gt0_rxdfelfhold_i;
    wire          gt0_rxlpmlfhold_i;
    wire          gt0_rxlpmhfhold_i;


    wire          gt1_txresetdone_i;
    wire          gt1_rxresetdone_i;
    wire          gt1_gttxreset_i;
    wire          gt1_gttxreset_t;
    wire          gt1_gtrxreset_i;
    wire          gt1_gtrxreset_t;
    wire          gt1_rxdfelpmreset_i;
    wire   [1:0]   gt1_txsysclksel_i;
    wire   [1:0]   gt1_rxsysclksel_i;
    wire          gt1_txuserrdy_i;
    wire          gt1_txuserrdy_t;
    wire          gt1_rxuserrdy_i;
    wire          gt1_rxuserrdy_t;

    wire          gt1_rxdfeagchold_i;
    wire          gt1_rxdfelfhold_i;
    wire          gt1_rxlpmlfhold_i;
    wire          gt1_rxlpmhfhold_i;
```

```
    wire        gt2_txresetdone_i;
    wire        gt2_rxresetdone_i;
    wire        gt2_gttxreset_i;
    wire        gt2_gttxreset_t;
    wire        gt2_gtrxreset_i;
    wire        gt2_gtrxreset_t;
    wire        gt2_rxdfelpmreset_i;
    wire  [1:0] gt2_txsysclksel_i;
    wire  [1:0] gt2_rxsysclksel_i;
    wire        gt2_txuserrdy_i;
    wire        gt2_txuserrdy_t;
    wire        gt2_rxuserrdy_i;
    wire        gt2_rxuserrdy_t;

    wire        gt2_rxdfeagchold_i;
    wire        gt2_rxdfelfhold_i;
    wire        gt2_rxlpmlfhold_i;
    wire        gt2_rxlpmhfhold_i;


    wire        gt3_txresetdone_i;
    wire        gt3_rxresetdone_i;
    wire        gt3_gttxreset_i;
    wire        gt3_gttxreset_t;
    wire        gt3_gtrxreset_i;
    wire        gt3_gtrxreset_t;
    wire        gt3_rxdfelpmreset_i;
    wire  [1:0] gt3_txsysclksel_i;
    wire  [1:0] gt3_rxsysclksel_i;
    wire        gt3_txuserrdy_i;
    wire        gt3_txuserrdy_t;
    wire        gt3_rxuserrdy_i;
    wire        gt3_rxuserrdy_t;

    wire        gt3_rxdfeagchold_i;
    wire        gt3_rxdfelfhold_i;
    wire        gt3_rxlpmlfhold_i;
    wire        gt3_rxlpmhfhold_i;




    wire        gt0_qpllreset_i;
    wire        gt0_qpllreset_t;
    wire        gt0_qpllrefclklost_i;
    wire        gt0_qpllock_i;


//------------------------------ Global Signals -----------------------------
    wire        tied_to_ground_i;
    wire        tied_to_vcc_i;

    wire        gt0_rxoutclk_i;
    wire        gt0_recclk_stable_i;

    wire        gt1_rxoutclk_i;
    wire        gt1_recclk_stable_i;

    wire        gt2_rxoutclk_i;
    wire        gt2_recclk_stable_i;

    wire        gt3_rxoutclk_i;
    wire        gt3_recclk_stable_i;

    integer  rx_cdrlock_counter= 0;
    reg          rx_cdrlocked;
```

```
//*************************** Main Body of Code ********************************
   //  Static signal Assigments
   assign  tied_to_ground_i                 =  1'b0;
   assign  tied_to_vcc_i                    =  1'b1;

//    ---------------------------- The GT Wrapper ----------------------------

   // Use the instantiation template in the example directory to add the GT wrapper to your design.
   // In this example, the wrapper is wired up for basic operation with a frame generator and frame
   // checker. The GTs will reset, then attempt to align and transmit data. If channel bonding is
   // enabled, bonding should occur after alignment.


   gtx8_chan_pll #
   (
      .WRAPPER_SIM_GTRESET_SPEEDUP    (EXAMPLE_SIM_GTRESET_SPEEDUP)
   )
   gtx8_chan_pll_i
   (

      //____________________________________________________________________________
      //____________________________________________________________________________
      //GT0  (X1Y12)
      .GT0_GTREFCLK0_IN              (gt0_gtrefclk0_in),
      //------------------------- Channel - DRP Ports  --------------------------
      .GT0_DRPADDR_IN                (GT0_DRPADDR_IN),
      .GT0_DRPCLK_IN                 (GT0_DRPCLK_IN),
      .GT0_DRPDI_IN                  (GT0_DRPDI_IN),
      .GT0_DRPDO_OUT                 (GT0_DRPDO_OUT),
      .GT0_DRPEN_IN                  (GT0_DRPEN_IN),
      .GT0_DRPRDY_OUT                (GT0_DRPRDY_OUT),
      .GT0_DRPWE_IN                  (GT0_DRPWE_IN),

      .GT0_RXOUTCLKSEL_IN            (GT0_RXOUTCLKSEL_IN),
      .GT0_TXOUTCLKSEL_IN            (GT0_TXOUTCLKSEL_IN),
      .GT0_TX8B10BEN_IN              (GT0_TX8B10BEN_IN),
      .GT0_RX8B10BEN_IN              (GT0_RX8B10BEN_IN),

      //------------------------------ Clocking Ports -----------------------------
      .GT0_RXSYSCLKSEL_IN            (GT0_RXSYSCLKSEL_IN),
      .GT0_TXSYSCLKSEL_IN            (GT0_TXSYSCLKSEL_IN),
      //------------------------------ Loopback Ports -----------------------------
      .GT0_LOOPBACK_IN               (GT0_LOOPBACK_IN),
      //-------------------------- PCI Express Ports ----------------------------
      .GT0_RXRATE_IN                 (GT0_RXRATE_IN),
      //------------------- RX Initialization and Reset Ports --------------------
      .GT0_RXUSERRDY_IN              (gt0_rxuserrdy_i),
      //------------------------ RX Margin Analysis Ports ------------------------
      .GT0_EYESCANDATAERROR_OUT      (GT0_EYESCANDATAERROR_OUT),
      //----------------------- Receive Ports - CDR Ports ------------------------
      .GT0_RXCDRLOCK_OUT             (GT0_RXCDRLOCK_OUT),
      //----------------- Receive Ports - FPGA RX Interface Ports -----------------
      .GT0_RXUSRCLK_IN               (GT0_RXUSRCLK_IN),
      .GT0_RXUSRCLK2_IN              (GT0_RXUSRCLK2_IN),
      //----------------- Receive Ports - FPGA RX interface Ports -----------------
      .GT0_RXDATA_OUT                (GT0_RXDATA_OUT),
      //----------------- Receive Ports - Pattern Checker Ports ------------------
      .GT0_RXPRBSERR_OUT             (GT0_RXPRBSERR_OUT),
      .GT0_RXPRBSSEL_IN              (GT0_RXPRBSSEL_IN),
      //----------------- Receive Ports - Pattern Checker ports ------------------
      .GT0_RXPRBSCNTRESET_IN         (GT0_RXPRBSCNTRESET_IN),
      //---------------------- Receive Ports - RX AFE -----------------------------
      .GT0_GTXRXP_IN                 (GT0_GTXRXP_IN),
      //---------------------- Receive Ports - RX AFE Ports -----------------------
      .GT0_GTXRXN_IN                 (GT0_GTXRXN_IN),
      //------------------- Receive Ports - RX Equalizer Ports --------------------
```

```
.GT0_RXDFEAGCHOLD_IN          (gt0_rxdfeagchold_i),
.GT0_RXDFELFHOLD_IN           (gt0_rxdfelfhold_i),
//---------- Receive Ports - RX Fabric ClocK Output Control Ports ----------
.GT0_RXRATEDONE_OUT           (GT0_RXRATEDONE_OUT),
//------------- Receive Ports - RX Fabric Output Control Ports -------------
.GT0_RXOUTCLK_OUT             (GT0_RXOUTCLK_OUT),
//------------ Receive Ports - RX Initialization and Reset Ports ------------
.GT0_GTRXRESET_IN             (gt0_gtrxreset_i),
.GT0_RXPMARESET_IN            (GT0_RXPMARESET_IN),
//---------------- Receive Ports - RX Polarity Control Ports ----------------
.GT0_RXPOLARITY_IN            (GT0_RXPOLARITY_IN),
//--------------------- Receive Ports - RX gearbox ports ---------------------
.GT0_RXSLIDE_IN               (GT0_RXSLIDE_IN),
//------------ Receive Ports -RX Initialization and Reset Ports ------------
.GT0_RXRESETDONE_OUT          (gt0_rxresetdone_i),
//----------------------- TX Configurable Driver Ports -----------------------
.GT0_TXPOSTCURSOR_IN          (GT0_TXPOSTCURSOR_IN),
.GT0_TXPRECURSOR_IN           (GT0_TXPRECURSOR_IN),
//------------------- TX Initialization and Reset Ports --------------------
.GT0_GTTXRESET_IN             (gt0_gttxreset_i),
.GT0_TXUSERRDY_IN             (gt0_txuserrdy_i),
//---------------- Transmit Ports - FPGA TX Interface Ports ----------------
.GT0_TXUSRCLK_IN              (GT0_TXUSRCLK_IN),
.GT0_TXUSRCLK2_IN             (GT0_TXUSRCLK2_IN),
//------------------- Transmit Ports - PCI Express Ports -------------------
.GT0_TXRATE_IN                (GT0_TXRATE_IN),
//---------------- Transmit Ports - Pattern Generator Ports ----------------
.GT0_TXPRBSFORCEERR_IN        (GT0_TXPRBSFORCEERR_IN),
//------------- Transmit Ports - TX Configurable Driver Ports --------------
.GT0_TXDIFFCTRL_IN            (GT0_TXDIFFCTRL_IN),
.GT0_TXMAINCURSOR_IN          (GT0_TXMAINCURSOR_IN),
//---------------- Transmit Ports - TX Data Path interface -----------------
.GT0_TXDATA_IN                (GT0_TXDATA_IN),
//-------------- Transmit Ports - TX Driver and OOB signaling --------------
.GT0_GTXTXN_OUT               (GT0_GTXTXN_OUT),
.GT0_GTXTXP_OUT               (GT0_GTXTXP_OUT),
//--------- Transmit Ports - TX Fabric Clock Output Control Ports ----------
.GT0_TXOUTCLK_OUT             (GT0_TXOUTCLK_OUT),
.GT0_TXOUTCLKFABRIC_OUT       (GT0_TXOUTCLKFABRIC_OUT),
.GT0_TXOUTCLKPCS_OUT          (GT0_TXOUTCLKPCS_OUT),
.GT0_TXRATEDONE_OUT           (GT0_TXRATEDONE_OUT),
//------------ Transmit Ports - TX Initialization and Reset Ports ------------
.GT0_TXRESETDONE_OUT          (gt0_txresetdone_i),
//---------------- Transmit Ports - TX Polarity Control Ports ----------------
.GT0_TXPOLARITY_IN            (GT0_TXPOLARITY_IN),
//---------------- Transmit Ports - pattern Generator Ports ----------------
.GT0_TXPRBSSEL_IN             (3'b011),
.GT0_TXPD_IN                  (GT0_TXPD_IN),



//____________________________________________________________________________
//____________________________________________________________________________
//GT1  (X1Y13)
.GT1_GTREFCLK0_IN             (gt1_gtrefclk0_in),
//-------------------------- Channel - DRP Ports  --------------------------
.GT1_DRPADDR_IN               (GT1_DRPADDR_IN),
.GT1_DRPCLK_IN                (GT1_DRPCLK_IN),
.GT1_DRPDI_IN                 (GT1_DRPDI_IN),
.GT1_DRPDO_OUT                (GT1_DRPDO_OUT),
.GT1_DRPEN_IN                 (GT1_DRPEN_IN),
.GT1_DRPRDY_OUT               (GT1_DRPRDY_OUT),
.GT1_DRPWE_IN                 (GT1_DRPWE_IN),

.GT1_RXOUTCLKSEL_IN           (GT1_RXOUTCLKSEL_IN),
.GT1_TXOUTCLKSEL_IN           (GT1_TXOUTCLKSEL_IN),
.GT1_TX8B10BEN_IN             (GT1_TX8B10BEN_IN),
.GT1_RX8B10BEN_IN             (GT1_RX8B10BEN_IN),

//------------------------------ Clocking Ports -----------------------------
```

```
.GT1_RXSYSCLKSEL_IN            (GT1_RXSYSCLKSEL_IN),
.GT1_TXSYSCLKSEL_IN            (GT1_TXSYSCLKSEL_IN),
//---------------------------- Loopback Ports -----------------------------
.GT1_LOOPBACK_IN               (GT1_LOOPBACK_IN),
//--------------------------- PCI Express Ports ----------------------------
.GT1_RXRATE_IN                 (GT1_RXRATE_IN),
//------------------- RX Initialization and Reset Ports --------------------
.GT1_RXUSERRDY_IN              (gt1_rxuserrdy_i),
//----------------------- RX Margin Analysis Ports ------------------------
.GT1_EYESCANDATAERROR_OUT      (GT1_EYESCANDATAERROR_OUT),
//----------------------- Receive Ports - CDR Ports ------------------------
.GT1_RXCDRLOCK_OUT             (GT1_RXCDRLOCK_OUT),
//---------------- Receive Ports - FPGA RX Interface Ports -----------------
.GT1_RXUSRCLK_IN               (GT1_RXUSRCLK_IN),
.GT1_RXUSRCLK2_IN              (GT1_RXUSRCLK2_IN),
//---------------- Receive Ports - FPGA RX interface Ports -----------------
.GT1_RXDATA_OUT                (GT1_RXDATA_OUT),
//----------------- Receive Ports - Pattern Checker Ports ------------------
.GT1_RXPRBSERR_OUT             (GT1_RXPRBSERR_OUT),
.GT1_RXPRBSSEL_IN              (GT1_RXPRBSSEL_IN),
//----------------- Receive Ports - Pattern Checker ports ------------------
.GT1_RXPRBSCNTRESET_IN         (GT1_RXPRBSCNTRESET_IN),
//------------------------ Receive Ports - RX AFE -------------------------
.GT1_GTXRXP_IN                 (GT1_GTXRXP_IN),
//---------------------- Receive Ports - RX AFE Ports ----------------------
.GT1_GTXRXN_IN                 (GT1_GTXRXN_IN),
//------------------- Receive Ports - RX Equalizer Ports -------------------
.GT1_RXDFEAGCHOLD_IN           (gt1_rxdfeagchold_i),
.GT1_RXDFELFHOLD_IN            (gt1_rxdfelfhold_i),
//---------- Receive Ports - RX Fabric ClocK Output Control Ports ----------
.GT1_RXRATEDONE_OUT            (GT1_RXRATEDONE_OUT),
//------------- Receive Ports - RX Fabric Output Control Ports -------------
.GT1_RXOUTCLK_OUT              (GT1_RXOUTCLK_OUT),
//----------- Receive Ports - RX Initialization and Reset Ports ------------
.GT1_GTRXRESET_IN              (gt1_gtrxreset_i),
.GT1_RXPMARESET_IN             (GT1_RXPMARESET_IN),
//---------------- Receive Ports - RX Polarity Control Ports ---------------
.GT1_RXPOLARITY_IN             (GT1_RXPOLARITY_IN),
//--------------------- Receive Ports - RX gearbox ports --------------------
.GT1_RXSLIDE_IN                (GT1_RXSLIDE_IN),
//------------ Receive Ports -RX Initialization and Reset Ports ------------
.GT1_RXRESETDONE_OUT           (gt1_rxresetdone_i),
//---------------------- TX Configurable Driver Ports ----------------------
.GT1_TXPOSTCURSOR_IN           (GT1_TXPOSTCURSOR_IN),
.GT1_TXPRECURSOR_IN            (GT1_TXPRECURSOR_IN),
//------------------- TX Initialization and Reset Ports --------------------
.GT1_GTTXRESET_IN              (gt1_gttxreset_i),
.GT1_TXUSERRDY_IN              (gt1_txuserrdy_i),
//---------------- Transmit Ports - FPGA TX Interface Ports ----------------
.GT1_TXUSRCLK_IN               (GT1_TXUSRCLK_IN),
.GT1_TXUSRCLK2_IN              (GT1_TXUSRCLK2_IN),
//------------------- Transmit Ports - PCI Express Ports -------------------
.GT1_TXRATE_IN                 (GT1_TXRATE_IN),
//---------------- Transmit Ports - Pattern Generator Ports ----------------
.GT1_TXPRBSFORCEERR_IN         (GT1_TXPRBSFORCEERR_IN),
//------------- Transmit Ports - TX Configurable Driver Ports --------------
.GT1_TXDIFFCTRL_IN             (GT1_TXDIFFCTRL_IN),
.GT1_TXMAINCURSOR_IN           (GT1_TXMAINCURSOR_IN),
//---------------- Transmit Ports - TX Data Path interface -----------------
.GT1_TXDATA_IN                 (GT1_TXDATA_IN),
//-------------- Transmit Ports - TX Driver and OOB signaling --------------
.GT1_GTXTXN_OUT                (GT1_GTXTXN_OUT),
.GT1_GTXTXP_OUT                (GT1_GTXTXP_OUT),
//--------- Transmit Ports - TX Fabric Clock Output Control Ports ----------
.GT1_TXOUTCLK_OUT              (GT1_TXOUTCLK_OUT),
.GT1_TXOUTCLKFABRIC_OUT        (GT1_TXOUTCLKFABRIC_OUT),
.GT1_TXOUTCLKPCS_OUT           (GT1_TXOUTCLKPCS_OUT),
.GT1_TXRATEDONE_OUT            (GT1_TXRATEDONE_OUT),
//----------- Transmit Ports - TX Initialization and Reset Ports -----------
.GT1_TXRESETDONE_OUT           (gt1_txresetdone_i),
```

```
//---------- Transmit Ports - TX Polarity Control Ports ----------
.GT1_TXPOLARITY_IN              (GT1_TXPOLARITY_IN),
//---------- Transmit Ports - pattern Generator Ports ----------
.GT1_TXPRBSSEL_IN               (3'b011),
.GT1_TXPD_IN                    (GT1_TXPD_IN),



//____________________________________________________________________________
//____________________________________________________________________________
//GT2  (X1Y14)
.GT2_GTREFCLK0_IN               (gt2_gtrefclk0_in),
//------------------------- Channel - DRP Ports  -------------------------
.GT2_DRPADDR_IN                 (GT2_DRPADDR_IN),
.GT2_DRPCLK_IN                  (GT2_DRPCLK_IN),
.GT2_DRPDI_IN                   (GT2_DRPDI_IN),
.GT2_DRPDO_OUT                  (GT2_DRPDO_OUT),
.GT2_DRPEN_IN                   (GT2_DRPEN_IN),
.GT2_DRPRDY_OUT                 (GT2_DRPRDY_OUT),
.GT2_DRPWE_IN                   (GT2_DRPWE_IN),

.GT2_RXOUTCLKSEL_IN             (GT2_RXOUTCLKSEL_IN),
.GT2_TXOUTCLKSEL_IN             (GT2_TXOUTCLKSEL_IN),
.GT2_TX8B10BEN_IN               (GT2_TX8B10BEN_IN),
.GT2_RX8B10BEN_IN               (GT2_RX8B10BEN_IN),

//------------------------------ Clocking Ports ------------------------------
.GT2_RXSYSCLKSEL_IN             (GT2_RXSYSCLKSEL_IN),
.GT2_TXSYSCLKSEL_IN             (GT2_TXSYSCLKSEL_IN),
//------------------------------ Loopback Ports ------------------------------
.GT2_LOOPBACK_IN                (GT2_LOOPBACK_IN),
//---------------------------- PCI Express Ports ----------------------------
.GT2_RXRATE_IN                  (GT2_RXRATE_IN),
//-------------------- RX Initialization and Reset Ports --------------------
.GT2_RXUSERRDY_IN               (gt2_rxuserrdy_i),
//------------------------ RX Margin Analysis Ports ------------------------
.GT2_EYESCANDATAERROR_OUT       (GT2_EYESCANDATAERROR_OUT),
//----------------------- Receive Ports - CDR Ports -----------------------
.GT2_RXCDRLOCK_OUT              (GT2_RXCDRLOCK_OUT),
//----------------- Receive Ports - FPGA RX Interface Ports -----------------
.GT2_RXUSRCLK_IN                (GT2_RXUSRCLK_IN),
.GT2_RXUSRCLK2_IN               (GT2_RXUSRCLK2_IN),
//----------------- Receive Ports - FPGA RX interface Ports -----------------
.GT2_RXDATA_OUT                 (GT2_RXDATA_OUT),
//----------------- Receive Ports - Pattern Checker Ports ------------------
.GT2_RXPRBSERR_OUT              (GT2_RXPRBSERR_OUT),
.GT2_RXPRBSSEL_IN               (GT2_RXPRBSSEL_IN),
//----------------- Receive Ports - Pattern Checker ports ------------------
.GT2_RXPRBSCNTRESET_IN          (GT2_RXPRBSCNTRESET_IN),
//------------------------ Receive Ports - RX AFE ------------------------
.GT2_GTXRXP_IN                  (GT2_GTXRXP_IN),
//---------------------- Receive Ports - RX AFE Ports ----------------------
.GT2_GTXRXN_IN                  (GT2_GTXRXN_IN),
//------------------- Receive Ports - RX Equalizer Ports -------------------
.GT2_RXDFEAGCHOLD_IN            (gt2_rxdfeagchold_i),
.GT2_RXDFELFHOLD_IN             (gt2_rxdfelfhold_i),
//---------- Receive Ports - RX Fabric ClocK Output Control Ports ----------
.GT2_RXRATEDONE_OUT             (GT2_RXRATEDONE_OUT),
//------------- Receive Ports - RX Fabric Output Control Ports -------------
.GT2_RXOUTCLK_OUT               (GT2_RXOUTCLK_OUT),
//----------- Receive Ports - RX Initialization and Reset Ports ------------
.GT2_GTRXRESET_IN               (gt2_gtrxreset_i),
.GT2_RXPMARESET_IN              (GT2_RXPMARESET_IN),
//---------------- Receive Ports - RX Polarity Control Ports ----------------
.GT2_RXPOLARITY_IN              (GT2_RXPOLARITY_IN),
//-------------------- Receive Ports - RX gearbox ports --------------------
.GT2_RXSLIDE_IN                 (GT2_RXSLIDE_IN),
//------------ Receive Ports -RX Initialization and Reset Ports ------------
.GT2_RXRESETDONE_OUT            (gt2_rxresetdone_i),
//---------------------- TX Configurable Driver Ports ----------------------
```

```
.GT2_TXPOSTCURSOR_IN          (GT2_TXPOSTCURSOR_IN),
.GT2_TXPRECURSOR_IN           (GT2_TXPRECURSOR_IN),
//------------------- TX Initialization and Reset Ports --------------------
.GT2_GTTXRESET_IN             (gt2_gttxreset_i),
.GT2_TXUSERRDY_IN             (gt2_txuserrdy_i),
//---------------- Transmit Ports - FPGA TX Interface Ports ----------------
.GT2_TXUSRCLK_IN              (GT2_TXUSRCLK_IN),
.GT2_TXUSRCLK2_IN             (GT2_TXUSRCLK2_IN),
//------------------ Transmit Ports - PCI Express Ports -------------------
.GT2_TXRATE_IN                (GT2_TXRATE_IN),
//---------------- Transmit Ports - Pattern Generator Ports ----------------
.GT2_TXPRBSFORCEERR_IN        (GT2_TXPRBSFORCEERR_IN),
//------------- Transmit Ports - TX Configurable Driver Ports --------------
.GT2_TXDIFFCTRL_IN            (GT2_TXDIFFCTRL_IN),
.GT2_TXMAINCURSOR_IN          (GT2_TXMAINCURSOR_IN),
//---------------- Transmit Ports - TX Data Path interface -----------------
.GT2_TXDATA_IN                (GT2_TXDATA_IN),
//-------------- Transmit Ports - TX Driver and OOB signaling --------------
.GT2_GTXTXN_OUT               (GT2_GTXTXN_OUT),
.GT2_GTXTXP_OUT               (GT2_GTXTXP_OUT),
//--------- Transmit Ports - TX Fabric Clock Output Control Ports ----------
.GT2_TXOUTCLK_OUT             (GT2_TXOUTCLK_OUT),
.GT2_TXOUTCLKFABRIC_OUT       (GT2_TXOUTCLKFABRIC_OUT),
.GT2_TXOUTCLKPCS_OUT          (GT2_TXOUTCLKPCS_OUT),
.GT2_TXRATEDONE_OUT           (GT2_TXRATEDONE_OUT),
//----------- Transmit Ports - TX Initialization and Reset Ports -----------
.GT2_TXRESETDONE_OUT          (gt2_txresetdone_i),
//---------------- Transmit Ports - TX Polarity Control Ports ----------------
.GT2_TXPOLARITY_IN            (GT2_TXPOLARITY_IN),
//---------------- Transmit Ports - pattern Generator Ports ----------------
.GT2_TXPRBSSEL_IN             (3'b011),
.GT2_TXPD_IN                  (GT2_TXPD_IN),



//____________________________________________________________________________
//____________________________________________________________________________
//GT3  (X1Y15)
.GT3_GTREFCLK0_IN             (gt3_gtrefclk0_in),
//--------------------------- Channel - DRP Ports  --------------------------
.GT3_DRPADDR_IN               (GT3_DRPADDR_IN),
.GT3_DRPCLK_IN                (GT3_DRPCLK_IN),
.GT3_DRPDI_IN                 (GT3_DRPDI_IN),
.GT3_DRPDO_OUT                (GT3_DRPDO_OUT),
.GT3_DRPEN_IN                 (GT3_DRPEN_IN),
.GT3_DRPRDY_OUT               (GT3_DRPRDY_OUT),
.GT3_DRPWE_IN                 (GT3_DRPWE_IN),

.GT3_RXOUTCLKSEL_IN           (GT3_RXOUTCLKSEL_IN),
.GT3_TXOUTCLKSEL_IN           (GT3_TXOUTCLKSEL_IN),
.GT3_TX8B10BEN_IN             (GT3_TX8B10BEN_IN),
.GT3_RX8B10BEN_IN             (GT3_RX8B10BEN_IN),

//------------------------------ Clocking Ports -----------------------------
.GT3_RXSYSCLKSEL_IN           (GT3_RXSYSCLKSEL_IN),
.GT3_TXSYSCLKSEL_IN           (GT3_TXSYSCLKSEL_IN),
//---------------------------- Loopback Ports ---------------------------
.GT3_LOOPBACK_IN              (GT3_LOOPBACK_IN),
//-------------------------- PCI Express Ports ----------------------------
.GT3_RXRATE_IN                (GT3_RXRATE_IN),
//------------------- RX Initialization and Reset Ports --------------------
.GT3_RXUSERRDY_IN             (gt3_rxuserrdy_i),
//------------------------ RX Margin Analysis Ports ------------------------
.GT3_EYESCANDATAERROR_OUT     (GT3_EYESCANDATAERROR_OUT),
//----------------------- Receive Ports - CDR Ports ------------------------
.GT3_RXCDRLOCK_OUT            (GT3_RXCDRLOCK_OUT),
//---------------- Receive Ports - FPGA RX Interface Ports -----------------
.GT3_RXUSRCLK_IN              (GT3_RXUSRCLK_IN),
.GT3_RXUSRCLK2_IN             (GT3_RXUSRCLK2_IN),
//---------------- Receive Ports - FPGA RX interface Ports -----------------
```

```
    .GT3_RXDATA_OUT                 (GT3_RXDATA_OUT),
    //---------------- Receive Ports - Pattern Checker Ports ------------------
    .GT3_RXPRBSERR_OUT              (GT3_RXPRBSERR_OUT),
    .GT3_RXPRBSSEL_IN               (GT3_RXPRBSSEL_IN),
    //---------------- Receive Ports - Pattern Checker ports ------------------
    .GT3_RXPRBSCNTRESET_IN          (GT3_RXPRBSCNTRESET_IN),
    //------------------------ Receive Ports - RX AFE -------------------------
    .GT3_GTXRXP_IN                  (GT3_GTXRXP_IN),
    //---------------------- Receive Ports - RX AFE Ports ----------------------
    .GT3_GTXRXN_IN                  (GT3_GTXRXN_IN),
    //------------------- Receive Ports - RX Equalizer Ports -------------------
    .GT3_RXDFEAGCHOLD_IN            (gt3_rxdfeagchold_i),
    .GT3_RXDFELFHOLD_IN             (gt3_rxdfelfhold_i),
    //---------- Receive Ports - RX Fabric ClocK Output Control Ports ----------
    .GT3_RXRATEDONE_OUT             (GT3_RXRATEDONE_OUT),
    //------------- Receive Ports - RX Fabric Output Control Ports -------------
    .GT3_RXOUTCLK_OUT               (GT3_RXOUTCLK_OUT),
    //----------- Receive Ports - RX Initialization and Reset Ports ------------
    .GT3_GTRXRESET_IN               (gt3_gtrxreset_i),
    .GT3_RXPMARESET_IN              (GT3_RXPMARESET_IN),
    //--------------- Receive Ports - RX Polarity Control Ports ----------------
    .GT3_RXPOLARITY_IN              (GT3_RXPOLARITY_IN),
    //--------------------- Receive Ports - RX gearbox ports ---------------------
    .GT3_RXSLIDE_IN                 (GT3_RXSLIDE_IN),
    //------------ Receive Ports -RX Initialization and Reset Ports ------------
    .GT3_RXRESETDONE_OUT            (gt3_rxresetdone_i),
    //---------------------- TX Configurable Driver Ports ----------------------
    .GT3_TXPOSTCURSOR_IN            (GT3_TXPOSTCURSOR_IN),
    .GT3_TXPRECURSOR_IN             (GT3_TXPRECURSOR_IN),
    //------------------- TX Initialization and Reset Ports --------------------
    .GT3_GTTXRESET_IN               (gt3_gttxreset_i),
    .GT3_TXUSERRDY_IN               (gt3_txuserrdy_i),
    //---------------- Transmit Ports - FPGA TX Interface Ports ----------------
    .GT3_TXUSRCLK_IN                (GT3_TXUSRCLK_IN),
    .GT3_TXUSRCLK2_IN               (GT3_TXUSRCLK2_IN),
    //------------------- Transmit Ports - PCI Express Ports -------------------
    .GT3_TXRATE_IN                  (GT3_TXRATE_IN),
    //---------------- Transmit Ports - Pattern Generator Ports ----------------
    .GT3_TXPRBSFORCEERR_IN          (GT3_TXPRBSFORCEERR_IN),
    //------------- Transmit Ports - TX Configurable Driver Ports --------------
    .GT3_TXDIFFCTRL_IN              (GT3_TXDIFFCTRL_IN),
    .GT3_TXMAINCURSOR_IN            (GT3_TXMAINCURSOR_IN),
    //---------------- Transmit Ports - TX Data Path interface -----------------
    .GT3_TXDATA_IN                  (GT3_TXDATA_IN),
    //-------------- Transmit Ports - TX Driver and OOB signaling --------------
    .GT3_GTXTXN_OUT                 (GT3_GTXTXN_OUT),
    .GT3_GTXTXP_OUT                 (GT3_GTXTXP_OUT),
    //--------- Transmit Ports - TX Fabric Clock Output Control Ports ----------
    .GT3_TXOUTCLK_OUT               (GT3_TXOUTCLK_OUT),
    .GT3_TXOUTCLKFABRIC_OUT         (GT3_TXOUTCLKFABRIC_OUT),
    .GT3_TXOUTCLKPCS_OUT            (GT3_TXOUTCLKPCS_OUT),
    .GT3_TXRATEDONE_OUT             (GT3_TXRATEDONE_OUT),
    //----------- Transmit Ports - TX Initialization and Reset Ports -----------
    .GT3_TXRESETDONE_OUT            (gt3_txresetdone_i),
    //---------------- Transmit Ports - TX Polarity Control Ports ----------------
    .GT3_TXPOLARITY_IN              (GT3_TXPOLARITY_IN),
    //---------------- Transmit Ports - pattern Generator Ports ----------------
    .GT3_TXPRBSSEL_IN               (3'b011),
    .GT3_TXPD_IN                    (GT3_TXPD_IN),


//____________________________________COMMON PORTS_____________________________________
    //---------------------- Common Block  - Ref Clock Ports ---------------------
    .GT0_GTREFCLK0_COMMON_IN        (GT0_GTREFCLK0_COMMON_IN),
    //----------------------- Common Block - QPLL Ports ------------------------
    .GT0_QPLLLOCK_OUT               (gt0_qpllock_i),
    .GT0_QPLLLOCKDETCLK_IN          (GT0_QPLLLOCKDETCLK_IN),
    .GT0_QPLLREFCLKLOST_OUT         (gt0_qpllrefclklost_i),
    .GT0_QPLLRESET_IN               (gt0_qpllreset_i),
```

```
    .GTC0_DRPADDR               (GTC0_DRPADDR),
    .GTC0_DRPCLK                (GTC0_DRPCLK),
    .GTC0_DRPDI                 (GTC0_DRPDI),
    .GTC0_DRPDO                 (GTC0_DRPDO),
    .GTC0_DRPEN                 (GTC0_DRPEN),
    .GTC0_DRPRDY                (GTC0_DRPRDY),
    .GTC0_DRPWE                 (GTC0_DRPWE)

);


assign  gt0_rxdfelpmreset_i             =  tied_to_ground_i;
assign  gt1_rxdfelpmreset_i             =  tied_to_ground_i;
assign  gt2_rxdfelpmreset_i             =  tied_to_ground_i;
assign  gt3_rxdfelpmreset_i             =  tied_to_ground_i;

assign  gt0_rxsysclksel_i               =  2'b11;
assign  gt0_txsysclksel_i               =  2'b11;
assign  gt1_rxsysclksel_i               =  2'b11;
assign  gt1_txsysclksel_i               =  2'b11;
assign  gt2_rxsysclksel_i               =  2'b11;
assign  gt2_txsysclksel_i               =  2'b11;
assign  gt3_rxsysclksel_i               =  2'b11;
assign  gt3_txsysclksel_i               =  2'b11;

assign  GT0_TXRESETDONE_OUT             =  gt0_txresetdone_i;
assign  GT0_RXRESETDONE_OUT             =  gt0_rxresetdone_i;
assign  GT1_TXRESETDONE_OUT             =  gt1_txresetdone_i;
assign  GT1_RXRESETDONE_OUT             =  gt1_rxresetdone_i;
assign  GT2_TXRESETDONE_OUT             =  gt2_txresetdone_i;
assign  GT2_RXRESETDONE_OUT             =  gt2_rxresetdone_i;
assign  GT3_TXRESETDONE_OUT             =  gt3_txresetdone_i;
assign  GT3_RXRESETDONE_OUT             =  gt3_rxresetdone_i;

assign  GT0_QPLLLOCK_OUT                =  gt0_qpllock_i;

generate
if (EXAMPLE_USE_CHIPSCOPE == 1)
begin : chipscope
assign  gt0_gttxreset_i                 =  GT0_GTTXRESET_IN || gt0_gttxreset_t;
assign  gt0_gtrxreset_i                 =  GT0_GTRXRESET_IN || gt0_gtrxreset_t;
assign  gt0_txuserrdy_i                 =  GT0_TXUSERRDY_IN || gt0_txuserrdy_t;
assign  gt0_rxuserrdy_i                 =  GT0_RXUSERRDY_IN || gt0_rxuserrdy_t;
assign  gt1_gttxreset_i                 =  GT1_GTTXRESET_IN || gt1_gttxreset_t;
assign  gt1_gtrxreset_i                 =  GT1_GTRXRESET_IN || gt1_gtrxreset_t;
assign  gt1_txuserrdy_i                 =  GT1_TXUSERRDY_IN || gt1_txuserrdy_t;
assign  gt1_rxuserrdy_i                 =  GT1_RXUSERRDY_IN || gt1_rxuserrdy_t;
assign  gt2_gttxreset_i                 =  GT2_GTTXRESET_IN || gt2_gttxreset_t;
assign  gt2_gtrxreset_i                 =  GT2_GTRXRESET_IN || gt2_gtrxreset_t;
assign  gt2_txuserrdy_i                 =  GT2_TXUSERRDY_IN || gt2_txuserrdy_t;
assign  gt2_rxuserrdy_i                 =  GT2_RXUSERRDY_IN || gt2_rxuserrdy_t;
assign  gt3_gttxreset_i                 =  GT3_GTTXRESET_IN || gt3_gttxreset_t;
assign  gt3_gtrxreset_i                 =  GT3_GTRXRESET_IN || gt3_gtrxreset_t;
assign  gt3_txuserrdy_i                 =  GT3_TXUSERRDY_IN || gt3_txuserrdy_t;
assign  gt3_rxuserrdy_i                 =  GT3_RXUSERRDY_IN || gt3_rxuserrdy_t;

assign  gt0_qpllreset_i                 =  GT0_QPLLRESET_IN || gt0_qpllreset_t;
end
endgenerate

generate
if (EXAMPLE_USE_CHIPSCOPE == 0)
begin : no_chipscope
assign  gt0_gttxreset_i                 =  gt0_gttxreset_t;
assign  gt0_gtrxreset_i                 =  gt0_gtrxreset_t;
assign  gt0_txuserrdy_i                 =  gt0_txuserrdy_t;
assign  gt0_rxuserrdy_i                 =  gt0_rxuserrdy_t;
assign  gt1_gttxreset_i                 =  gt1_gttxreset_t;
assign  gt1_gtrxreset_i                 =  gt1_gtrxreset_t;
assign  gt1_txuserrdy_i                 =  gt1_txuserrdy_t;
```

```
    assign  gt1_rxuserrdy_i                 =  gt1_rxuserrdy_t;
    assign  gt2_gttxreset_i                 =  gt2_gttxreset_t;
    assign  gt2_gtrxreset_i                 =  gt2_gtrxreset_t;
    assign  gt2_txuserrdy_i                 =  gt2_txuserrdy_t;
    assign  gt2_rxuserrdy_i                 =  gt2_rxuserrdy_t;
    assign  gt3_gttxreset_i                 =  gt3_gttxreset_t;
    assign  gt3_gtrxreset_i                 =  gt3_gtrxreset_t;
    assign  gt3_txuserrdy_i                 =  gt3_txuserrdy_t;
    assign  gt3_rxuserrdy_i                 =  gt3_rxuserrdy_t;

    assign  gt0_qpllreset_i                 =  gt0_qpllreset_t;
  end
  endgenerate


  gtx8_chan_pll_TX_STARTUP_FSM #
        (
         .GT_TYPE               ("GTX"), //GTX or GTH or GTP
         .STABLE_CLOCK_PERIOD     (STABLE_CLOCK_PERIOD),         // Period of the stable clock driving this state-machine,
unit is [ns]
         .RETRY_COUNTER_BITWIDTH   (8),
         .TX_QPLL_USED          ("TRUE"),                    // the TX and RX Reset FSMs must
         .RX_QPLL_USED          ("TRUE"),                    // share these two generic values
         .PHASE_ALIGNMENT_MANUAL  ("FALSE")          // Decision if a manual phase-alignment is necessary or the automatic
                                                     // is enough. For single-lane applications the automatic alignment is
                                                     // sufficient
          )
  gt0_txresetfsm_i
          (
        .STABLE_CLOCK               (SYSCLK_IN),
        .TXUSERCLK                  (GT0_TXUSRCLK_IN),
        .SOFT_RESET                 (SOFT_RESET_IN),
        .QPLLREFCLKLOST               (gt0_qpllrefclklost_i),
        .CPLLREFCLKLOST               (tied_to_ground_i),
        .QPLLLOCK                   (gt0_qpllock_i),
        .CPLLLOCK                   (tied_to_vcc_i),
        .TXRESETDONE                (gt0_txresetdone_i),
        .MMCM_LOCK                  (tied_to_vcc_i),
        .GTTXRESET                  (gt0_gttxreset_t),
        .MMCM_RESET                 (),
        .QPLL_RESET                 (gt0_qpllreset_t),
        .CPLL_RESET                 (),
        .TX_FSM_RESET_DONE            (GT0_TX_FSM_RESET_DONE_OUT),
        .TXUSERRDY                  (gt0_txuserrdy_t),
        .RUN_PHALIGNMENT              (),
        .RESET_PHALIGNMENT            (),
        .PHALIGNMENT_DONE             (tied_to_vcc_i),
        .RETRY_COUNTER              ()
          );


  gtx8_chan_pll_TX_STARTUP_FSM #
        (
         .GT_TYPE               ("GTX"), //GTX or GTH or GTP
         .STABLE_CLOCK_PERIOD     (STABLE_CLOCK_PERIOD),         // Period of the stable clock driving this state-machine,
unit is [ns]
         .RETRY_COUNTER_BITWIDTH   (8),
         .TX_QPLL_USED          ("TRUE"),                    // the TX and RX Reset FSMs must
         .RX_QPLL_USED          ("TRUE"),                    // share these two generic values
         .PHASE_ALIGNMENT_MANUAL  ("FALSE")          // Decision if a manual phase-alignment is necessary or the automatic
                                                     // is enough. For single-lane applications the automatic alignment is
                                                     // sufficient
          )
  gt1_txresetfsm_i
          (
        .STABLE_CLOCK               (SYSCLK_IN),
        .TXUSERCLK                  (GT1_TXUSRCLK_IN),
        .SOFT_RESET                 (SOFT_RESET_IN),
        .QPLLREFCLKLOST               (gt0_qpllrefclklost_i),
        .CPLLREFCLKLOST               (tied_to_ground_i),
```

```
        .QPLLLOCK                   (gt0_qpllock_i),
        .CPLLLOCK                   (tied_to_vcc_i),
        .TXRESETDONE                (gt1_txresetdone_i),
        .MMCM_LOCK                  (tied_to_vcc_i),
        .GTTXRESET                  (gt1_gttxreset_t),
        .MMCM_RESET                 (),
        .QPLL_RESET                 (),
        .CPLL_RESET                 (),
        .TX_FSM_RESET_DONE          (GT1_TX_FSM_RESET_DONE_OUT),
        .TXUSERRDY                  (gt1_txuserrdy_t),
        .RUN_PHALIGNMENT            (),
        .RESET_PHALIGNMENT          (),
        .PHALIGNMENT_DONE           (tied_to_vcc_i),
        .RETRY_COUNTER              ()
          );


   gtx8_chan_pll_TX_STARTUP_FSM #
          (
           .GT_TYPE              ("GTX"), //GTX or GTH or GTP
           .STABLE_CLOCK_PERIOD     (STABLE_CLOCK_PERIOD),         // Period of the stable clock driving this state-machine,
unit is [ns]
           .RETRY_COUNTER_BITWIDTH   (8),
           .TX_QPLL_USED          ("TRUE"),                   // the TX and RX Reset FSMs must
           .RX_QPLL_USED          ("TRUE"),                   // share these two generic values
           .PHASE_ALIGNMENT_MANUAL  ("FALSE")          // Decision if a manual phase-alignment is necessary or the automatic
                                                       // is enough. For single-lane applications the automatic alignment is
                                                       // sufficient
            )
   gt2_txresetfsm_i
           (
        .STABLE_CLOCK               (SYSCLK_IN),
        .TXUSERCLK                  (GT2_TXUSRCLK_IN),
        .SOFT_RESET                 (SOFT_RESET_IN),
        .QPLLREFCLKLOST             (gt0_qpllrefclklost_i),
        .CPLLREFCLKLOST             (tied_to_ground_i),
        .QPLLLOCK                   (gt0_qpllock_i),
        .CPLLLOCK                   (tied_to_vcc_i),
        .TXRESETDONE                (gt2_txresetdone_i),
        .MMCM_LOCK                  (tied_to_vcc_i),
        .GTTXRESET                  (gt2_gttxreset_t),
        .MMCM_RESET                 (),
        .QPLL_RESET                 (),
        .CPLL_RESET                 (),
        .TX_FSM_RESET_DONE          (GT2_TX_FSM_RESET_DONE_OUT),
        .TXUSERRDY                  (gt2_txuserrdy_t),
        .RUN_PHALIGNMENT            (),
        .RESET_PHALIGNMENT          (),
        .PHALIGNMENT_DONE           (tied_to_vcc_i),
        .RETRY_COUNTER              ()
          );


   gtx8_chan_pll_TX_STARTUP_FSM #
          (
           .GT_TYPE              ("GTX"), //GTX or GTH or GTP
           .STABLE_CLOCK_PERIOD     (STABLE_CLOCK_PERIOD),         // Period of the stable clock driving this state-machine,
unit is [ns]
           .RETRY_COUNTER_BITWIDTH   (8),
           .TX_QPLL_USED          ("TRUE"),                   // the TX and RX Reset FSMs must
           .RX_QPLL_USED          ("TRUE"),                   // share these two generic values
           .PHASE_ALIGNMENT_MANUAL  ("FALSE")          // Decision if a manual phase-alignment is necessary or the automatic
                                                       // is enough. For single-lane applications the automatic alignment is
                                                       // sufficient
            )
   gt3_txresetfsm_i
           (
        .STABLE_CLOCK               (SYSCLK_IN),
        .TXUSERCLK                  (GT3_TXUSRCLK_IN),
        .SOFT_RESET                 (SOFT_RESET_IN),
```

```
        .QPLLREFCLKLOST             (gt0_qpllrefclklost_i),
        .CPLLREFCLKLOST             (tied_to_ground_i),
        .QPLLLOCK                   (gt0_qpllock_i),
        .CPLLLOCK                   (tied_to_vcc_i),
        .TXRESETDONE                (gt3_txresetdone_i),
        .MMCM_LOCK                  (tied_to_vcc_i),
        .GTTXRESET                  (gt3_gttxreset_t),
        .MMCM_RESET                 (),
        .QPLL_RESET                 (),
        .CPLL_RESET                 (),
        .TX_FSM_RESET_DONE          (GT3_TX_FSM_RESET_DONE_OUT),
        .TXUSERRDY                  (gt3_txuserrdy_t),
        .RUN_PHALIGNMENT            (),
        .RESET_PHALIGNMENT          (),
        .PHALIGNMENT_DONE           (tied_to_vcc_i),
        .RETRY_COUNTER              ()
          );



   gtx8_chan_pll_RX_STARTUP_FSM  #
          (
           .EXAMPLE_SIMULATION      (EXAMPLE_SIMULATION),
           .GT_TYPE                 ("GTX"), //GTX or GTH or GTP
           .EQ_MODE                 ("DFE"),              //Rx Equalization Mode - Set to DFE or LPM
           .STABLE_CLOCK_PERIOD     (STABLE_CLOCK_PERIOD),          //Period of the stable clock driving this state-machine,
unit is [ns]
           .RETRY_COUNTER_BITWIDTH   (8),
           .TX_QPLL_USED            ("TRUE"),                   // the TX and RX Reset FSMs must
           .RX_QPLL_USED            ("TRUE"),                   // share these two generic values
           .PHASE_ALIGNMENT_MANUAL   ("FALSE")                  // Decision if a manual phase-alignment is necessary or the
automatic
                                                               // is enough. For single-lane applications the automatic alignment is
                                                               // sufficient
            )
   gt0_rxresetfsm_i
            (
        .STABLE_CLOCK               (SYSCLK_IN),
        .RXUSERCLK                  (GT0_RXUSRCLK_IN),
        .SOFT_RESET                 (SOFT_RESET_IN),
        .DONT_RESET_ON_DATA_ERROR      (DONT_RESET_ON_DATA_ERROR_IN),
        .QPLLREFCLKLOST             (gt0_qpllrefclklost_i),
        .CPLLREFCLKLOST             (tied_to_ground_i),
        .QPLLLOCK                   (gt0_qpllock_i),
        .CPLLLOCK                   (tied_to_vcc_i),
        .RXRESETDONE                (gt0_rxresetdone_i),
        .MMCM_LOCK                  (tied_to_vcc_i),
        .RECCLK_STABLE              (gt0_recclk_stable_i),
        .RECCLK_MONITOR_RESTART       (tied_to_ground_i),
        .DATA_VALID                 (GT0_DATA_VALID_IN),
        .TXUSERRDY                  (gt0_txuserrdy_i),
        .GTRXRESET                  (gt0_gtrxreset_t),
        .MMCM_RESET                 (),
        .QPLL_RESET                 (),
        .CPLL_RESET                 (),
        .RX_FSM_RESET_DONE          (GT0_RX_FSM_RESET_DONE_OUT),
        .RXUSERRDY                  (gt0_rxuserrdy_t),
        .RUN_PHALIGNMENT            (),
        .RESET_PHALIGNMENT          (),
        .PHALIGNMENT_DONE           (tied_to_vcc_i),
        .RXDFEAGCHOLD               (gt0_rxdfeagchold_i),
        .RXDFELFHOLD                (gt0_rxdfelfhold_i),
        .RXLPMLFHOLD                (gt0_rxlpmlfhold_i),
        .RXLPMHFHOLD                (gt0_rxlpmhfhold_i),
        .RETRY_COUNTER              ()
          );

   gtx8_chan_pll_RX_STARTUP_FSM  #
          (
           .EXAMPLE_SIMULATION      (EXAMPLE_SIMULATION),
```

```
            .GT_TYPE                ("GTX"), //GTX or GTH or GTP
            .EQ_MODE                ("DFE"),                //Rx Equalization Mode - Set to DFE or LPM
            .STABLE_CLOCK_PERIOD    (STABLE_CLOCK_PERIOD),          //Period of the stable clock driving this state-machine,
unit is [ns]
            .RETRY_COUNTER_BITWIDTH   (8),
            .TX_QPLL_USED           ("TRUE"),                       // the TX and RX Reset FSMs must
            .RX_QPLL_USED           ("TRUE"),                       // share these two generic values
            .PHASE_ALIGNMENT_MANUAL   ("FALSE")                 // Decision if a manual phase-alignment is necessary or the
automatic
                                                    // is enough. For single-lane applications the automatic alignment is
                                                    // sufficient
             )
    gt1_rxresetfsm_i
             (
           .STABLE_CLOCK                (SYSCLK_IN),
           .RXUSERCLK                   (GT1_RXUSRCLK_IN),
           .SOFT_RESET                  (SOFT_RESET_IN),
           .DONT_RESET_ON_DATA_ERROR      (DONT_RESET_ON_DATA_ERROR_IN),
           .QPLLREFCLKLOST              (gt0_qpllrefclklost_i),
           .CPLLREFCLKLOST              (tied_to_ground_i),
           .QPLLLOCK                    (gt0_qpllock_i),
           .CPLLLOCK                    (tied_to_vcc_i),
           .RXRESETDONE                 (gt1_rxresetdone_i),
           .MMCM_LOCK                   (tied_to_vcc_i),
           .RECCLK_STABLE               (gt1_recclk_stable_i),
           .RECCLK_MONITOR_RESTART      (tied_to_ground_i),
           .DATA_VALID                  (GT1_DATA_VALID_IN),
           .TXUSERRDY                   (gt1_txuserrdy_i),
           .GTRXRESET                   (gt1_gtrxreset_t),
           .MMCM_RESET                  (),
           .QPLL_RESET                  (),
           .CPLL_RESET                  (),
           .RX_FSM_RESET_DONE           (GT1_RX_FSM_RESET_DONE_OUT),
           .RXUSERRDY                   (gt1_rxuserrdy_t),
           .RUN_PHALIGNMENT             (),
           .RESET_PHALIGNMENT           (),
           .PHALIGNMENT_DONE            (tied_to_vcc_i),
           .RXDFEAGCHOLD                (gt1_rxdfeagchold_i),
           .RXDFELFHOLD                 (gt1_rxdfelfhold_i),
           .RXLPMLFHOLD                 (gt1_rxlpmlfhold_i),
           .RXLPMHFHOLD                 (gt1_rxlpmhfhold_i),
           .RETRY_COUNTER               ()
            );

    gtx8_chan_pll_RX_STARTUP_FSM  #
           (
            .EXAMPLE_SIMULATION      (EXAMPLE_SIMULATION),
            .GT_TYPE                ("GTX"), //GTX or GTH or GTP
            .EQ_MODE                ("DFE"),                //Rx Equalization Mode - Set to DFE or LPM
            .STABLE_CLOCK_PERIOD    (STABLE_CLOCK_PERIOD),          //Period of the stable clock driving this state-machine,
unit is [ns]
            .RETRY_COUNTER_BITWIDTH   (8),
            .TX_QPLL_USED           ("TRUE"),                       // the TX and RX Reset FSMs must
            .RX_QPLL_USED           ("TRUE"),                       // share these two generic values
            .PHASE_ALIGNMENT_MANUAL   ("FALSE")                 // Decision if a manual phase-alignment is necessary or the
automatic
                                                    // is enough. For single-lane applications the automatic alignment is
                                                    // sufficient
             )
    gt2_rxresetfsm_i
             (
           .STABLE_CLOCK                (SYSCLK_IN),
           .RXUSERCLK                   (GT2_RXUSRCLK_IN),
           .SOFT_RESET                  (SOFT_RESET_IN),
           .DONT_RESET_ON_DATA_ERROR      (DONT_RESET_ON_DATA_ERROR_IN),
           .QPLLREFCLKLOST              (gt0_qpllrefclklost_i),
           .CPLLREFCLKLOST              (tied_to_ground_i),
           .QPLLLOCK                    (gt0_qpllock_i),
           .CPLLLOCK                    (tied_to_vcc_i),
           .RXRESETDONE                 (gt2_rxresetdone_i),
```

```
        .MMCM_LOCK                  (tied_to_vcc_i),
        .RECCLK_STABLE              (gt2_recclk_stable_i),
        .RECCLK_MONITOR_RESTART     (tied_to_ground_i),
        .DATA_VALID                 (GT2_DATA_VALID_IN),
        .TXUSERRDY                  (gt2_txuserrdy_i),
        .GTRXRESET                  (gt2_gtrxreset_t),
        .MMCM_RESET                 (),
        .QPLL_RESET                 (),
        .CPLL_RESET                 (),
        .RX_FSM_RESET_DONE          (GT2_RX_FSM_RESET_DONE_OUT),
        .RXUSERRDY                  (gt2_rxuserrdy_t),
        .RUN_PHALIGNMENT            (),
        .RESET_PHALIGNMENT          (),
        .PHALIGNMENT_DONE           (tied_to_vcc_i),
        .RXDFEAGCHOLD               (gt2_rxdfeagchold_i),
        .RXDFELFHOLD                (gt2_rxdfelfhold_i),
        .RXLPMLFHOLD                (gt2_rxlpmlfhold_i),
        .RXLPMHFHOLD                (gt2_rxlpmhfhold_i),
        .RETRY_COUNTER              ()
        );

  gtx8_chan_pll_RX_STARTUP_FSM #
        (
        .EXAMPLE_SIMULATION     (EXAMPLE_SIMULATION),
        .GT_TYPE                ("GTX"), //GTX or GTH or GTP
        .EQ_MODE                ("DFE"),            //Rx Equalization Mode - Set to DFE or LPM
        .STABLE_CLOCK_PERIOD    (STABLE_CLOCK_PERIOD),          //Period of the stable clock driving this state-machine,
unit is [ns]
        .RETRY_COUNTER_BITWIDTH  (8),
        .TX_QPLL_USED           ("TRUE"),                   // the TX and RX Reset FSMs must
        .RX_QPLL_USED           ("TRUE"),                   // share these two generic values
        .PHASE_ALIGNMENT_MANUAL  ("FALSE")                  // Decision if a manual phase-alignment is necessary or the
automatic
                                                            // is enough. For single-lane applications the automatic alignment is
                                                            // sufficient
        )
  gt3_rxresetfsm_i
        (
        .STABLE_CLOCK               (SYSCLK_IN),
        .RXUSERCLK                  (GT3_RXUSRCLK_IN),
        .SOFT_RESET                 (SOFT_RESET_IN),
        .DONT_RESET_ON_DATA_ERROR   (DONT_RESET_ON_DATA_ERROR_IN),
        .QPLLREFCLKLOST             (gt0_qpllrefclklost_i),
        .CPLLREFCLKLOST             (tied_to_ground_i),
        .QPLLLOCK                   (gt0_qpllock_i),
        .CPLLLOCK                   (tied_to_vcc_i),
        .RXRESETDONE                (gt3_rxresetdone_i),
        .MMCM_LOCK                  (tied_to_vcc_i),
        .RECCLK_STABLE              (gt3_recclk_stable_i),
        .RECCLK_MONITOR_RESTART     (tied_to_ground_i),
        .DATA_VALID                 (GT3_DATA_VALID_IN),
        .TXUSERRDY                  (gt3_txuserrdy_i),
        .GTRXRESET                  (gt3_gtrxreset_t),
        .MMCM_RESET                 (),
        .QPLL_RESET                 (),
        .CPLL_RESET                 (),
        .RX_FSM_RESET_DONE          (GT3_RX_FSM_RESET_DONE_OUT),
        .RXUSERRDY                  (gt3_rxuserrdy_t),
        .RUN_PHALIGNMENT            (),
        .RESET_PHALIGNMENT          (),
        .PHALIGNMENT_DONE           (tied_to_vcc_i),
        .RXDFEAGCHOLD               (gt3_rxdfeagchold_i),
        .RXDFELFHOLD                (gt3_rxdfelfhold_i),
        .RXLPMLFHOLD                (gt3_rxlpmlfhold_i),
        .RXLPMHFHOLD                (gt3_rxlpmhfhold_i),
        .RETRY_COUNTER              ()
        );


   always @(posedge SYSCLK_IN)
```

```
 begin
     if(gt0_gtrxreset_i)
     begin
       rx_cdrlocked      <= `DLY    1'b0;
       rx_cdrlock_counter <= `DLY    0;
     end
     else if (rx_cdrlock_counter == WAIT_TIME_CDRLOCK)
     begin
       rx_cdrlocked      <= `DLY    1'b1;
       rx_cdrlock_counter <= `DLY    rx_cdrlock_counter;
     end
     else
       rx_cdrlock_counter <= `DLY    rx_cdrlock_counter + 1;
 end

assign  gt0_recclk_stable_i              =  rx_cdrlocked;
assign  gt1_recclk_stable_i              =  rx_cdrlocked;
assign  gt2_recclk_stable_i              =  rx_cdrlocked;
assign  gt3_recclk_stable_i              =  rx_cdrlocked;

endmodule



//****************************** Entity Declaration ****************************

(* CORE_GENERATION_INFO = "gtx8_chan_pll,gtwizard_v2_7,{protocol_file=Start_from_scratch}" *) module gtx8_chan_pll #
(
   // Simulation attributes
   parameter   WRAPPER_SIM_GTRESET_SPEEDUP    =   "FALSE",     // Set to "true" to speed up sim reset
   parameter   RX_DFE_KL_CFG2_IN              =   32'h301148AC,
   parameter   PMA_RSV_IN                     =   32'h001E7080
)
(
   //____________________________________________________________________________
   //____________________________________________________________________________
   //GT0  (X0Y12)
   //____________________________CHANNEL PORTS____________________________________

   input          GT0_GTREFCLK0_IN,
   //-------------------------- Channel - DRP Ports  --------------------------
   input   [8:0]   GT0_DRPADDR_IN,
   input          GT0_DRPCLK_IN,
   input   [15:0]  GT0_DRPDI_IN,
   output  [15:0]  GT0_DRPDO_OUT,
   input          GT0_DRPEN_IN,
   output          GT0_DRPRDY_OUT,
   input          GT0_DRPWE_IN,

   input   [2:0]   GT0_RXOUTCLKSEL_IN,
   input   [2:0]   GT0_TXOUTCLKSEL_IN,
   input          GT0_TX8B10BEN_IN,
   input          GT0_RX8B10BEN_IN,

   //------------------------------ Clocking Ports ------------------------------
   input   [1:0]   GT0_RXSYSCLKSEL_IN,
   input   [1:0]   GT0_TXSYSCLKSEL_IN,
   //---------------------------- Loopback Ports ----------------------------
   input   [2:0]   GT0_LOOPBACK_IN,
   //---------------------------- PCI Express Ports ----------------------------
   input   [2:0]   GT0_RXRATE_IN,
   //------------------- RX Initialization and Reset Ports --------------------
   input          GT0_RXUSERRDY_IN,
   //------------------------ RX Margin Analysis Ports ------------------------
   output          GT0_EYESCANDATAERROR_OUT,
   //---------------------- Receive Ports - CDR Ports -----------------------
   output          GT0_RXCDRLOCK_OUT,
   //---------------- Receive Ports - FPGA RX Interface Ports -----------------
   input          GT0_RXUSRCLK_IN,
   input          GT0_RXUSRCLK2_IN,
```

```
//---------------- Receive Ports - FPGA RX interface Ports -----------------
output  [31:0]  GT0_RXDATA_OUT,
//---------------- Receive Ports - Pattern Checker Ports ------------------
output          GT0_RXPRBSERR_OUT,
input   [2:0]   GT0_RXPRBSSEL_IN,
//---------------- Receive Ports - Pattern Checker ports ------------------
input           GT0_RXPRBSCNTRESET_IN,
//------------------------ Receive Ports - RX AFE -------------------------
input           GT0_GTXRXP_IN,
//---------------------- Receive Ports - RX AFE Ports ----------------------
input           GT0_GTXRXN_IN,
//------------------- Receive Ports - RX Equalizer Ports -------------------
input           GT0_RXDFEAGCHOLD_IN,
input           GT0_RXDFELFHOLD_IN,
//---------- Receive Ports - RX Fabric ClocK Output Control Ports ----------
output          GT0_RXRATEDONE_OUT,
//------------- Receive Ports - RX Fabric Output Control Ports -------------
output          GT0_RXOUTCLK_OUT,
//----------- Receive Ports - RX Initialization and Reset Ports ------------
input           GT0_GTRXRESET_IN,
input           GT0_RXPMARESET_IN,
//---------------- Receive Ports - RX Polarity Control Ports ----------------
input           GT0_RXPOLARITY_IN,
//--------------------- Receive Ports - RX gearbox ports --------------------
input           GT0_RXSLIDE_IN,
//------------- Receive Ports -RX Initialization and Reset Ports ------------
output          GT0_RXRESETDONE_OUT,
//---------------------- TX Configurable Driver Ports ----------------------
input   [4:0]   GT0_TXPOSTCURSOR_IN,
input   [4:0]   GT0_TXPRECURSOR_IN,
//------------------- TX Initialization and Reset Ports --------------------
input           GT0_GTTXRESET_IN,
input           GT0_TXUSERRDY_IN,
//---------------- Transmit Ports - FPGA TX Interface Ports ----------------
input           GT0_TXUSRCLK_IN,
input           GT0_TXUSRCLK2_IN,
//------------------- Transmit Ports - PCI Express Ports -------------------
input   [2:0]   GT0_TXRATE_IN,
//---------------- Transmit Ports - Pattern Generator Ports ----------------
input           GT0_TXPRBSFORCEERR_IN,
//------------- Transmit Ports - TX Configurable Driver Ports --------------
input   [3:0]   GT0_TXDIFFCTRL_IN,
input   [6:0]   GT0_TXMAINCURSOR_IN,
//---------------- Transmit Ports - TX Data Path interface -----------------
input   [31:0]  GT0_TXDATA_IN,
//-------------- Transmit Ports - TX Driver and OOB signaling --------------
output          GT0_GTXTXN_OUT,
output          GT0_GTXTXP_OUT,
//--------- Transmit Ports - TX Fabric Clock Output Control Ports ----------
output          GT0_TXOUTCLK_OUT,
output          GT0_TXOUTCLKFABRIC_OUT,
output          GT0_TXOUTCLKPCS_OUT,
output          GT0_TXRATEDONE_OUT,
//----------- Transmit Ports - TX Initialization and Reset Ports -----------
output          GT0_TXRESETDONE_OUT,
//---------------- Transmit Ports - TX Polarity Control Ports ----------------
input           GT0_TXPOLARITY_IN,
//---------------- Transmit Ports - pattern Generator Ports ----------------
input   [2:0]   GT0_TXPRBSSEL_IN,
input   [1:0]   GT0_TXPD_IN,

//____________________________________________________________________________
//____________________________________________________________________________
//GT1  (X0Y13)
//_____________________________CHANNEL PORTS_____________________________
input           GT1_GTREFCLK0_IN,
//------------------------- Channel - DRP Ports  --------------------------
input   [8:0]   GT1_DRPADDR_IN,
input           GT1_DRPCLK_IN,
input   [15:0]  GT1_DRPDI_IN,
```

```
output  [15:0]  GT1_DRPDO_OUT,
input           GT1_DRPEN_IN,
output          GT1_DRPRDY_OUT,
input           GT1_DRPWE_IN,

input   [2:0]   GT1_RXOUTCLKSEL_IN,
input   [2:0]   GT1_TXOUTCLKSEL_IN,
input           GT1_TX8B10BEN_IN,
input           GT1_RX8B10BEN_IN,

//------------------------------ Clocking Ports -----------------------------
input   [1:0]   GT1_RXSYSCLKSEL_IN,
input   [1:0]   GT1_TXSYSCLKSEL_IN,
//------------------------------ Loopback Ports -----------------------------
input   [2:0]   GT1_LOOPBACK_IN,
//-------------------------- PCI Express Ports ----------------------------
input   [2:0]   GT1_RXRATE_IN,
//------------------- RX Initialization and Reset Ports --------------------
input           GT1_RXUSERRDY_IN,
//------------------------ RX Margin Analysis Ports ------------------------
output          GT1_EYESCANDATAERROR_OUT,
//----------------------- Receive Ports - CDR Ports -------------------------
output          GT1_RXCDRLOCK_OUT,
//----------------- Receive Ports - FPGA RX Interface Ports -----------------
input           GT1_RXUSRCLK_IN,
input           GT1_RXUSRCLK2_IN,
//----------------- Receive Ports - FPGA RX interface Ports -----------------
output  [31:0]  GT1_RXDATA_OUT,
//----------------- Receive Ports - Pattern Checker Ports ------------------
output          GT1_RXPRBSERR_OUT,
input   [2:0]   GT1_RXPRBSSEL_IN,
//----------------- Receive Ports - Pattern Checker ports ------------------
input           GT1_RXPRBSCNTRESET_IN,
//------------------------ Receive Ports - RX AFE -------------------------
input           GT1_GTXRXP_IN,
//---------------------- Receive Ports - RX AFE Ports ----------------------
input           GT1_GTXRXN_IN,
//------------------- Receive Ports - RX Equalizer Ports -------------------
input           GT1_RXDFEAGCHOLD_IN,
input           GT1_RXDFELFHOLD_IN,
//---------- Receive Ports - RX Fabric ClocK Output Control Ports ----------
output          GT1_RXRATEDONE_OUT,
//------------- Receive Ports - RX Fabric Output Control Ports -------------
output          GT1_RXOUTCLK_OUT,
//----------- Receive Ports - RX Initialization and Reset Ports ------------
input           GT1_GTRXRESET_IN,
input           GT1_RXPMARESET_IN,
//---------------- Receive Ports - RX Polarity Control Ports ---------------
input           GT1_RXPOLARITY_IN,
//-------------------- Receive Ports - RX gearbox ports --------------------
input           GT1_RXSLIDE_IN,
//------------ Receive Ports -RX Initialization and Reset Ports ------------
output          GT1_RXRESETDONE_OUT,
//---------------------- TX Configurable Driver Ports ----------------------
input   [4:0]   GT1_TXPOSTCURSOR_IN,
input   [4:0]   GT1_TXPRECURSOR_IN,
//------------------- TX Initialization and Reset Ports --------------------
input           GT1_GTTXRESET_IN,
input           GT1_TXUSERRDY_IN,
//----------------- Transmit Ports - FPGA TX Interface Ports -----------------
input           GT1_TXUSRCLK_IN,
input           GT1_TXUSRCLK2_IN,
//------------------- Transmit Ports - PCI Express Ports -------------------
input   [2:0]   GT1_TXRATE_IN,
//----------------- Transmit Ports - Pattern Generator Ports -----------------
input           GT1_TXPRBSFORCEERR_IN,
//------------- Transmit Ports - TX Configurable Driver Ports --------------
input   [3:0]   GT1_TXDIFFCTRL_IN,
input   [6:0]   GT1_TXMAINCURSOR_IN,
//----------------- Transmit Ports - TX Data Path interface ------------------
```

```
input   [31:0]  GT1_TXDATA_IN,
//-------------- Transmit Ports - TX Driver and OOB signaling --------------
output          GT1_GTXTXN_OUT,
output          GT1_GTXTXP_OUT,
//---------- Transmit Ports - TX Fabric Clock Output Control Ports ----------
output          GT1_TXOUTCLK_OUT,
output          GT1_TXOUTCLKFABRIC_OUT,
output          GT1_TXOUTCLKPCS_OUT,
output          GT1_TXRATEDONE_OUT,
//------------ Transmit Ports - TX Initialization and Reset Ports -----------
output          GT1_TXRESETDONE_OUT,
//---------------- Transmit Ports - TX Polarity Control Ports ----------------
input           GT1_TXPOLARITY_IN,
//------------------ Transmit Ports - pattern Generator Ports ----------------
input   [2:0]   GT1_TXPRBSSEL_IN,
input   [1:0]   GT1_TXPD_IN,

//________________________________________________________________________
//________________________________________________________________________
//GT2  (X0Y14)
//____________________________CHANNEL PORTS________________________________
input           GT2_GTREFCLK0_IN,
//--------------------------- Channel - DRP Ports  ---------------------------
input   [8:0]   GT2_DRPADDR_IN,
input           GT2_DRPCLK_IN,
input   [15:0]  GT2_DRPDI_IN,
output  [15:0]  GT2_DRPDO_OUT,
input           GT2_DRPEN_IN,
output          GT2_DRPRDY_OUT,
input           GT2_DRPWE_IN,

input   [2:0]   GT2_RXOUTCLKSEL_IN,
input   [2:0]   GT2_TXOUTCLKSEL_IN,
input           GT2_TX8B10BEN_IN,
input           GT2_RX8B10BEN_IN,

//------------------------------ Clocking Ports -----------------------------
input   [1:0]   GT2_RXSYSCLKSEL_IN,
input   [1:0]   GT2_TXSYSCLKSEL_IN,
//------------------------------ Loopback Ports -----------------------------
input   [2:0]   GT2_LOOPBACK_IN,
//--------------------------- PCI Express Ports ----------------------------
input   [2:0]   GT2_RXRATE_IN,
//-------------------- RX Initialization and Reset Ports --------------------
input           GT2_RXUSERRDY_IN,
//------------------------ RX Margin Analysis Ports ------------------------
output          GT2_EYESCANDATAERROR_OUT,
//----------------------- Receive Ports - CDR Ports -----------------------
output          GT2_RXCDRLOCK_OUT,
//----------------- Receive Ports - FPGA RX Interface Ports -----------------
input           GT2_RXUSRCLK_IN,
input           GT2_RXUSRCLK2_IN,
//----------------- Receive Ports - FPGA RX interface Ports -----------------
output  [31:0]  GT2_RXDATA_OUT,
//------------------ Receive Ports - Pattern Checker Ports ------------------
output          GT2_RXPRBSERR_OUT,
input   [2:0]   GT2_RXPRBSSEL_IN,
//------------------ Receive Ports - Pattern Checker ports ------------------
input           GT2_RXPRBSCNTRESET_IN,
//------------------------- Receive Ports - RX AFE -------------------------
input           GT2_GTXRXP_IN,
//---------------------- Receive Ports - RX AFE Ports ----------------------
input           GT2_GTXRXN_IN,
//-------------------- Receive Ports - RX Equalizer Ports --------------------
input           GT2_RXDFEAGCHOLD_IN,
input           GT2_RXDFELFHOLD_IN,
//---------- Receive Ports - RX Fabric ClocK Output Control Ports ----------
output          GT2_RXRATEDONE_OUT,
//------------- Receive Ports - RX Fabric Output Control Ports -------------
output          GT2_RXOUTCLK_OUT,
```

```
//----------- Receive Ports - RX Initialization and Reset Ports ------------
input          GT2_GTRXRESET_IN,
input          GT2_RXPMARESET_IN,
//---------------- Receive Ports - RX Polarity Control Ports ----------------
input          GT2_RXPOLARITY_IN,
//--------------------- Receive Ports - RX gearbox ports --------------------
input          GT2_RXSLIDE_IN,
//------------ Receive Ports -RX Initialization and Reset Ports ------------
output         GT2_RXRESETDONE_OUT,
//---------------------- TX Configurable Driver Ports ----------------------
input   [4:0]   GT2_TXPOSTCURSOR_IN,
input   [4:0]   GT2_TXPRECURSOR_IN,
//------------------- TX Initialization and Reset Ports --------------------
input          GT2_GTTXRESET_IN,
input          GT2_TXUSERRDY_IN,
//---------------- Transmit Ports - FPGA TX Interface Ports ----------------
input          GT2_TXUSRCLK_IN,
input          GT2_TXUSRCLK2_IN,
//------------------- Transmit Ports - PCI Express Ports -------------------
input   [2:0]   GT2_TXRATE_IN,
//---------------- Transmit Ports - Pattern Generator Ports ----------------
input          GT2_TXPRBSFORCEERR_IN,
//------------- Transmit Ports - TX Configurable Driver Ports --------------
input   [3:0]   GT2_TXDIFFCTRL_IN,
input   [6:0]   GT2_TXMAINCURSOR_IN,
//---------------- Transmit Ports - TX Data Path interface -----------------
input   [31:0]  GT2_TXDATA_IN,
//-------------- Transmit Ports - TX Driver and OOB signaling --------------
output         GT2_GTXTXN_OUT,
output         GT2_GTXTXP_OUT,
//--------- Transmit Ports - TX Fabric Clock Output Control Ports ----------
output         GT2_TXOUTCLK_OUT,
output         GT2_TXOUTCLKFABRIC_OUT,
output         GT2_TXOUTCLKPCS_OUT,
output         GT2_TXRATEDONE_OUT,
//----------- Transmit Ports - TX Initialization and Reset Ports -----------
output         GT2_TXRESETDONE_OUT,
//---------------- Transmit Ports - TX Polarity Control Ports ----------------
input          GT2_TXPOLARITY_IN,
//---------------- Transmit Ports - pattern Generator Ports ----------------
input   [2:0]   GT2_TXPRBSSEL_IN,
input   [1:0]   GT2_TXPD_IN,

//____________________________________________________________________________
//____________________________________________________________________________
//GT3  (X0Y15)
//_____________________________CHANNEL PORTS_____________________________

input          GT3_GTREFCLK0_IN,
//-------------------------- Channel - DRP Ports  --------------------------
input   [8:0]   GT3_DRPADDR_IN,
input          GT3_DRPCLK_IN,
input   [15:0]  GT3_DRPDI_IN,
output  [15:0]  GT3_DRPDO_OUT,
input          GT3_DRPEN_IN,
output         GT3_DRPRDY_OUT,
input          GT3_DRPWE_IN,

input   [2:0]   GT3_RXOUTCLKSEL_IN,
input   [2:0]   GT3_TXOUTCLKSEL_IN,
input          GT3_TX8B10BEN_IN,
input          GT3_RX8B10BEN_IN,

//------------------------------ Clocking Ports ------------------------------
input   [1:0]   GT3_RXSYSCLKSEL_IN,
input   [1:0]   GT3_TXSYSCLKSEL_IN,
//---------------------------- Loopback Ports ----------------------------
input   [2:0]   GT3_LOOPBACK_IN,
//-------------------------- PCI Express Ports ----------------------------
input   [2:0]   GT3_RXRATE_IN,
```

```
//------------------- RX Initialization and Reset Ports --------------------
input          GT3_RXUSERRDY_IN,
//------------------------ RX Margin Analysis Ports ------------------------
output         GT3_EYESCANDATAERROR_OUT,
//----------------------- Receive Ports - CDR Ports ------------------------
output         GT3_RXCDRLOCK_OUT,
//----------------- Receive Ports - FPGA RX Interface Ports ------------------
input          GT3_RXUSRCLK_IN,
input          GT3_RXUSRCLK2_IN,
//----------------- Receive Ports - FPGA RX interface Ports ------------------
output  [31:0]  GT3_RXDATA_OUT,
//----------------- Receive Ports - Pattern Checker Ports ------------------
output         GT3_RXPRBSERR_OUT,
input   [2:0]   GT3_RXPRBSSEL_IN,
//----------------- Receive Ports - Pattern Checker ports ------------------
input          GT3_RXPRBSCNTRESET_IN,
//------------------------ Receive Ports - RX AFE -------------------------
input          GT3_GTXRXP_IN,
//---------------------- Receive Ports - RX AFE Ports ----------------------
input          GT3_GTXRXN_IN,
//------------------- Receive Ports - RX Equalizer Ports -------------------
input          GT3_RXDFEAGCHOLD_IN,
input          GT3_RXDFELFHOLD_IN,
//---------- Receive Ports - RX Fabric ClocK Output Control Ports ----------
output         GT3_RXRATEDONE_OUT,
//------------- Receive Ports - RX Fabric Output Control Ports -------------
output         GT3_RXOUTCLK_OUT,
//----------- Receive Ports - RX Initialization and Reset Ports ------------
input          GT3_GTRXRESET_IN,
input          GT3_RXPMARESET_IN,
//--------------- Receive Ports - RX Polarity Control Ports ----------------
input          GT3_RXPOLARITY_IN,
//------------------- Receive Ports - RX gearbox ports --------------------
input          GT3_RXSLIDE_IN,
//------------ Receive Ports -RX Initialization and Reset Ports ------------
output         GT3_RXRESETDONE_OUT,
//---------------------- TX Configurable Driver Ports ----------------------
input   [4:0]   GT3_TXPOSTCURSOR_IN,
input   [4:0]   GT3_TXPRECURSOR_IN,
//------------------- TX Initialization and Reset Ports --------------------
input          GT3_GTTXRESET_IN,
input          GT3_TXUSERRDY_IN,
//----------------- Transmit Ports - FPGA TX Interface Ports -----------------
input          GT3_TXUSRCLK_IN,
input          GT3_TXUSRCLK2_IN,
//------------------- Transmit Ports - PCI Express Ports -------------------
input   [2:0]   GT3_TXRATE_IN,
//----------------- Transmit Ports - Pattern Generator Ports -----------------
input          GT3_TXPRBSFORCEERR_IN,
//------------- Transmit Ports - TX Configurable Driver Ports --------------
input   [3:0]   GT3_TXDIFFCTRL_IN,
input   [6:0]   GT3_TXMAINCURSOR_IN,
//----------------- Transmit Ports - TX Data Path interface -----------------
input   [31:0]  GT3_TXDATA_IN,
//-------------- Transmit Ports - TX Driver and OOB signaling --------------
output         GT3_GTXTXN_OUT,
output         GT3_GTXTXP_OUT,
//--------- Transmit Ports - TX Fabric Clock Output Control Ports ----------
output         GT3_TXOUTCLK_OUT,
output         GT3_TXOUTCLKFABRIC_OUT,
output         GT3_TXOUTCLKPCS_OUT,
output         GT3_TXRATEDONE_OUT,
//----------- Transmit Ports - TX Initialization and Reset Ports -----------
output         GT3_TXRESETDONE_OUT,
//---------------- Transmit Ports - TX Polarity Control Ports ----------------
input          GT3_TXPOLARITY_IN,
//----------------- Transmit Ports - pattern Generator Ports -----------------
input   [2:0]   GT3_TXPRBSSEL_IN,
input   [1:0]   GT3_TXPD_IN,
```

```
    //_____________________________COMMON PORTS_____________________________
    //---------------------- Common Block  - Ref Clock Ports ----------------------
    input          GT0_GTREFCLK0_COMMON_IN,
    //------------------------ Common Block - QPLL Ports ------------------------
    output          GT0_QPLLLOCK_OUT,
    input          GT0_QPLLLOCKDETCLK_IN,
    output          GT0_QPLLREFCLKLOST_OUT,
    input          GT0_QPLLRESET_IN,

    input   [7:0]   GTC0_DRPADDR,
    input          GTC0_DRPCLK,
    input  [15:0]   GTC0_DRPDI,
    output [15:0]   GTC0_DRPDO,
    input          GTC0_DRPEN,
    output          GTC0_DRPRDY,
    input          GTC0_DRPWE


);
//***************************** Parameter Declarations ************************
    parameter QPLL_FBDIV_TOP =  16;

    parameter QPLL_FBDIV_IN  =  (QPLL_FBDIV_TOP == 16)  ? 10'b0000100000 :
                                            (QPLL_FBDIV_TOP == 20)  ? 10'b0000110000 :
                                            (QPLL_FBDIV_TOP == 32)  ? 10'b0001100000 :
                                            (QPLL_FBDIV_TOP == 40)  ? 10'b0010000000 :
                                            (QPLL_FBDIV_TOP == 64)  ? 10'b0011100000 :
                                            (QPLL_FBDIV_TOP == 66)  ? 10'b0101000000 :
                                            (QPLL_FBDIV_TOP == 80)  ? 10'b0100100000 :
                                            (QPLL_FBDIV_TOP == 100) ? 10'b0101110000 : 10'b0000000000;

   parameter QPLL_FBDIV_RATIO = (QPLL_FBDIV_TOP == 16)  ? 1'b1 :
                                            (QPLL_FBDIV_TOP == 20)  ? 1'b1 :
                                            (QPLL_FBDIV_TOP == 32)  ? 1'b1 :
                                            (QPLL_FBDIV_TOP == 40)  ? 1'b1 :
                                            (QPLL_FBDIV_TOP == 64)  ? 1'b1 :
                                            (QPLL_FBDIV_TOP == 66)  ? 1'b0 :
                                            (QPLL_FBDIV_TOP == 80)  ? 1'b1 :
                                            (QPLL_FBDIV_TOP == 100) ? 1'b1 : 1'b1;

//***************************** Wire Declarations *****************************

    // ground and vcc signals
    wire           tied_to_ground_i;
    wire    [63:0]  tied_to_ground_vec_i;
    wire           tied_to_vcc_i;
    wire    [63:0]  tied_to_vcc_vec_i;

    wire           gt0_qploutclk_i;
    wire           gt0_qploutrefclk_i;


    wire           gt1_qploutclk_i;
    wire           gt1_qploutrefclk_i;

    wire           gt0_qpllclk_i;
    wire           gt0_qpllrefclk_i;

    wire           gt1_qpllclk_i;
    wire           gt1_qpllrefclk_i;

    wire           gt2_qpllclk_i;
    wire           gt2_qpllrefclk_i;

    wire           gt3_qpllclk_i;
    wire           gt3_qpllrefclk_i;
```

```
//*********************************** Main Body of Code**************************

    assign tied_to_ground_i             = 1'b0;
    assign tied_to_ground_vec_i         = 64'h0000000000000000;
    assign tied_to_vcc_i                = 1'b1;
    assign tied_to_vcc_vec_i            = 64'hffffffffffffffff;


    assign  gt0_qpllclk_i    = gt0_qplloutclk_i;
    assign  gt0_qpllrefclk_i = gt0_qplloutrefclk_i;
    assign  gt1_qpllclk_i    = gt0_qplloutclk_i;
    assign  gt1_qpllrefclk_i = gt0_qplloutrefclk_i;
    assign  gt2_qpllclk_i    = gt0_qplloutclk_i;
    assign  gt2_qpllrefclk_i = gt0_qplloutrefclk_i;
    assign  gt3_qpllclk_i    = gt0_qplloutclk_i;
    assign  gt3_qpllrefclk_i = gt0_qplloutrefclk_i;



//------------------------- GT Instances  -------------------------------
    //_____________________________________________________________________
    //_____________________________________________________________________
    //GT0  (X0Y12)

    gtx8_chan_pll_GT #
    (
        // Simulation attributes
        .GT_SIM_GTRESET_SPEEDUP   (WRAPPER_SIM_GTRESET_SPEEDUP),
        .RX_DFE_KL_CFG2_IN        (RX_DFE_KL_CFG2_IN),
        .PCS_RSVD_ATTR_IN         (48'h000000000000),
        .PMA_RSV_IN               (PMA_RSV_IN)
    )
    gt0_gtx8_chan_pll_i
    (
        //------------------------- Channel - Clocking Ports ------------------------
        .gtrefclk0_in                   (GT0_GTREFCLK0_IN),
        //-------------------------- Channel - DRP Ports  --------------------------
        .DRPADDR_IN                     (GT0_DRPADDR_IN),
        .DRPCLK_IN                      (GT0_DRPCLK_IN),
        .DRPDI_IN                       (GT0_DRPDI_IN),
        .DRPDO_OUT                      (GT0_DRPDO_OUT),
        .DRPEN_IN                       (GT0_DRPEN_IN),
        .DRPRDY_OUT                     (GT0_DRPRDY_OUT),
        .DRPWE_IN                       (GT0_DRPWE_IN),
        //------------------------------ Clocking Ports -----------------------------
        .QPLLCLK_IN                     (gt0_qpllclk_i),
        .QPLLREFCLK_IN                  (gt0_qpllrefclk_i),
        .RXSYSCLKSEL_IN                 (GT0_RXSYSCLKSEL_IN),
        .TXSYSCLKSEL_IN                 (GT0_TXSYSCLKSEL_IN),
        .RXOUTCLKSEL_IN                 (GT0_RXOUTCLKSEL_IN),
        .TXOUTCLKSEL_IN                 (GT0_TXOUTCLKSEL_IN),
        .RX8B10BEN_IN                   (GT0_RX8B10BEN_IN),
        .TX8B10BEN_IN                   (GT0_TX8B10BEN_IN),
        //------------------------------ Loopback Ports -----------------------------
        .LOOPBACK_IN                    (GT0_LOOPBACK_IN),
        //---------------------------- PCI Express Ports ----------------------------
        .RXRATE_IN                      (GT0_RXRATE_IN),
        //------------------- RX Initialization and Reset Ports --------------------
        .RXUSERRDY_IN                   (GT0_RXUSERRDY_IN),
        //------------------------ RX Margin Analysis Ports ------------------------
        .EYESCANDATAERROR_OUT           (GT0_EYESCANDATAERROR_OUT),
        //----------------------- Receive Ports - CDR Ports -----------------------
        .RXCDRLOCK_OUT                  (GT0_RXCDRLOCK_OUT),
        //----------------- Receive Ports - FPGA RX Interface Ports -----------------
        .RXUSRCLK_IN                    (GT0_RXUSRCLK_IN),
        .RXUSRCLK2_IN                   (GT0_RXUSRCLK2_IN),
        //----------------- Receive Ports - FPGA RX interface Ports -----------------
        .RXDATA_OUT                     (GT0_RXDATA_OUT),
        //----------------- Receive Ports - Pattern Checker Ports ------------------
        .RXPRBSERR_OUT                  (GT0_RXPRBSERR_OUT),
```

```
    .RXPRBSSEL_IN                (GT0_RXPRBSSEL_IN),
    //---------------- Receive Ports - Pattern Checker ports ------------------
    .RXPRBSCNTRESET_IN           (GT0_RXPRBSCNTRESET_IN),
    //------------------------ Receive Ports - RX AFE -------------------------
    .GTXRXP_IN                   (GT0_GTXRXP_IN),
    //---------------------- Receive Ports - RX AFE Ports ----------------------
    .GTXRXN_IN                   (GT0_GTXRXN_IN),
    //------------------- Receive Ports - RX Equalizer Ports -------------------
    .RXDFEAGCHOLD_IN             (GT0_RXDFEAGCHOLD_IN),
    .RXDFELFHOLD_IN              (GT0_RXDFELFHOLD_IN),
    //---------- Receive Ports - RX Fabric ClocK Output Control Ports ----------
    .RXRATEDONE_OUT              (GT0_RXRATEDONE_OUT),
    //------------- Receive Ports - RX Fabric Output Control Ports -------------
    .RXOUTCLK_OUT                (GT0_RXOUTCLK_OUT),
    //----------- Receive Ports - RX Initialization and Reset Ports ------------
    .GTRXRESET_IN                (GT0_GTRXRESET_IN),
    .RXPMARESET_IN               (GT0_RXPMARESET_IN),
    //---------------- Receive Ports - RX Polarity Control Ports ----------------
    .RXPOLARITY_IN               (GT0_RXPOLARITY_IN),
    //-------------------- Receive Ports - RX gearbox ports --------------------
    .RXSLIDE_IN                  (GT0_RXSLIDE_IN),
    //------------ Receive Ports -RX Initialization and Reset Ports ------------
    .RXRESETDONE_OUT             (GT0_RXRESETDONE_OUT),
    //---------------------- TX Configurable Driver Ports ----------------------
    .TXPOSTCURSOR_IN             (GT0_TXPOSTCURSOR_IN),
    .TXPRECURSOR_IN              (GT0_TXPRECURSOR_IN),
    //------------------- TX Initialization and Reset Ports --------------------
    .GTTXRESET_IN                (GT0_GTTXRESET_IN),
    .TXUSERRDY_IN                (GT0_TXUSERRDY_IN),
    //---------------- Transmit Ports - FPGA TX Interface Ports ----------------
    .TXUSRCLK_IN                 (GT0_TXUSRCLK_IN),
    .TXUSRCLK2_IN                (GT0_TXUSRCLK2_IN),
    //------------------ Transmit Ports - PCI Express Ports --------------------
    .TXRATE_IN                   (GT0_TXRATE_IN),
    //---------------- Transmit Ports - Pattern Generator Ports ----------------
    .TXPRBSFORCEERR_IN           (GT0_TXPRBSFORCEERR_IN),
    //------------- Transmit Ports - TX Configurable Driver Ports --------------
    .TXDIFFCTRL_IN               (GT0_TXDIFFCTRL_IN),
    .TXMAINCURSOR_IN             (GT0_TXMAINCURSOR_IN),
    //---------------- Transmit Ports - TX Data Path interface -----------------
    .TXDATA_IN                   (GT0_TXDATA_IN),
    //-------------- Transmit Ports - TX Driver and OOB signaling --------------
    .GTXTXN_OUT                  (GT0_GTXTXN_OUT),
    .GTXTXP_OUT                  (GT0_GTXTXP_OUT),
    //--------- Transmit Ports - TX Fabric Clock Output Control Ports ----------
    .TXOUTCLK_OUT                (GT0_TXOUTCLK_OUT),
    .TXOUTCLKFABRIC_OUT          (GT0_TXOUTCLKFABRIC_OUT),
    .TXOUTCLKPCS_OUT             (GT0_TXOUTCLKPCS_OUT),
    .TXRATEDONE_OUT              (GT0_TXRATEDONE_OUT),
    //----------- Transmit Ports - TX Initialization and Reset Ports -----------
    .TXRESETDONE_OUT             (GT0_TXRESETDONE_OUT),
    //---------------- Transmit Ports - TX Polarity Control Ports ----------------
    .TXPOLARITY_IN               (GT0_TXPOLARITY_IN),
    //---------------- Transmit Ports - pattern Generator Ports ----------------
    .TXPRBSSEL_IN                (GT0_TXPRBSSEL_IN),
    .TXPD_IN                     (GT0_TXPD_IN)

);

//___________________________________________________________________________
//___________________________________________________________________________
//GT1  (X0Y13)

gtx8_chan_pll_GT #
(
    // Simulation attributes
    .GT_SIM_GTRESET_SPEEDUP   (WRAPPER_SIM_GTRESET_SPEEDUP),
    .RX_DFE_KL_CFG2_IN        (RX_DFE_KL_CFG2_IN),
    .PCS_RSVD_ATTR_IN         (48'h000000000000),
    .PMA_RSV_IN               (PMA_RSV_IN)
```

```
)
gt1_gtx8_chan_pll_i
(
    //------------------------ Channel - Clocking Ports ------------------------
    .gtrefclk0_in                (GT1_GTREFCLK0_IN),
    //-------------------------- Channel - DRP Ports  --------------------------
    .DRPADDR_IN                   (GT1_DRPADDR_IN),
    .DRPCLK_IN                    (GT1_DRPCLK_IN),
    .DRPDI_IN                     (GT1_DRPDI_IN),
    .DRPDO_OUT                    (GT1_DRPDO_OUT),
    .DRPEN_IN                     (GT1_DRPEN_IN),
    .DRPRDY_OUT                   (GT1_DRPRDY_OUT),
    .DRPWE_IN                     (GT1_DRPWE_IN),
    //------------------------------ Clocking Ports ------------------------------
    .QPLLCLK_IN                   (gt1_qpllclk_i),
    .QPLLREFCLK_IN                (gt1_qpllrefclk_i),
    .RXSYSCLKSEL_IN               (GT1_RXSYSCLKSEL_IN),
    .TXSYSCLKSEL_IN               (GT1_TXSYSCLKSEL_IN),
    .RXOUTCLKSEL_IN               (GT1_RXOUTCLKSEL_IN),
    .TXOUTCLKSEL_IN               (GT1_TXOUTCLKSEL_IN),
    .RX8B10BEN_IN                 (GT1_RX8B10BEN_IN),
    .TX8B10BEN_IN                 (GT1_TX8B10BEN_IN),
    //------------------------------ Loopback Ports ------------------------------
    .LOOPBACK_IN                  (GT1_LOOPBACK_IN),
    //-------------------------- PCI Express Ports --------------------------
    .RXRATE_IN                    (GT1_RXRATE_IN),
    //------------------ RX Initialization and Reset Ports --------------------
    .RXUSERRDY_IN                 (GT1_RXUSERRDY_IN),
    //------------------------ RX Margin Analysis Ports ------------------------
    .EYESCANDATAERROR_OUT         (GT1_EYESCANDATAERROR_OUT),
    //------------------------ Receive Ports - CDR Ports ------------------------
    .RXCDRLOCK_OUT                (GT1_RXCDRLOCK_OUT),
    //---------------- Receive Ports - FPGA RX Interface Ports -----------------
    .RXUSRCLK_IN                  (GT1_RXUSRCLK_IN),
    .RXUSRCLK2_IN                 (GT1_RXUSRCLK2_IN),
    //---------------- Receive Ports - FPGA RX interface Ports -----------------
    .RXDATA_OUT                   (GT1_RXDATA_OUT),
    //------------------ Receive Ports - Pattern Checker Ports -------------------
    .RXPRBSERR_OUT                (GT1_RXPRBSERR_OUT),
    .RXPRBSSEL_IN                 (GT1_RXPRBSSEL_IN),
    //------------------ Receive Ports - Pattern Checker ports -------------------
    .RXPRBSCNTRESET_IN            (GT1_RXPRBSCNTRESET_IN),
    //------------------------ Receive Ports - RX AFE ------------------------
    .GTXRXP_IN                    (GT1_GTXRXP_IN),
    //---------------------- Receive Ports - RX AFE Ports ----------------------
    .GTXRXN_IN                    (GT1_GTXRXN_IN),
    //------------------- Receive Ports - RX Equalizer Ports -------------------
    .RXDFEAGCHOLD_IN              (GT1_RXDFEAGCHOLD_IN),
    .RXDFELFHOLD_IN               (GT1_RXDFELFHOLD_IN),
    //---------- Receive Ports - RX Fabric ClocK Output Control Ports ----------
    .RXRATEDONE_OUT               (GT1_RXRATEDONE_OUT),
    //------------- Receive Ports - RX Fabric Output Control Ports -------------
    .RXOUTCLK_OUT                 (GT1_RXOUTCLK_OUT),
    //----------- Receive Ports - RX Initialization and Reset Ports ------------
    .GTRXRESET_IN                 (GT1_GTRXRESET_IN),
    .RXPMARESET_IN                (GT1_RXPMARESET_IN),
    //---------------- Receive Ports - RX Polarity Control Ports ----------------
    .RXPOLARITY_IN                (GT1_RXPOLARITY_IN),
    //--------------------- Receive Ports - RX gearbox ports ---------------------
    .RXSLIDE_IN                   (GT1_RXSLIDE_IN),
    //------------ Receive Ports -RX Initialization and Reset Ports ------------
    .RXRESETDONE_OUT              (GT1_RXRESETDONE_OUT),
    //---------------------- TX Configurable Driver Ports ----------------------
    .TXPOSTCURSOR_IN              (GT1_TXPOSTCURSOR_IN),
    .TXPRECURSOR_IN               (GT1_TXPRECURSOR_IN),
    //------------------- TX Initialization and Reset Ports --------------------
    .GTTXRESET_IN                 (GT1_GTTXRESET_IN),
    .TXUSERRDY_IN                 (GT1_TXUSERRDY_IN),
    //---------------- Transmit Ports - FPGA TX Interface Ports ----------------
    .TXUSRCLK_IN                  (GT1_TXUSRCLK_IN),
```

```
    .TXUSRCLK2_IN                (GT1_TXUSRCLK2_IN),
    //------------------- Transmit Ports - PCI Express Ports -------------------
    .TXRATE_IN                   (GT1_TXRATE_IN),
    //---------------- Transmit Ports - Pattern Generator Ports ----------------
    .TXPRBSFORCEERR_IN           (GT1_TXPRBSFORCEERR_IN),
    //-------------- Transmit Ports - TX Configurable Driver Ports --------------
    .TXDIFFCTRL_IN               (GT1_TXDIFFCTRL_IN),
    .TXMAINCURSOR_IN             (GT1_TXMAINCURSOR_IN),
    //---------------- Transmit Ports - TX Data Path interface -----------------
    .TXDATA_IN                   (GT1_TXDATA_IN),
    //-------------- Transmit Ports - TX Driver and OOB signaling --------------
    .GTXTXN_OUT                  (GT1_GTXTXN_OUT),
    .GTXTXP_OUT                  (GT1_GTXTXP_OUT),
    //--------- Transmit Ports - TX Fabric Clock Output Control Ports ----------
    .TXOUTCLK_OUT                (GT1_TXOUTCLK_OUT),
    .TXOUTCLKFABRIC_OUT          (GT1_TXOUTCLKFABRIC_OUT),
    .TXOUTCLKPCS_OUT             (GT1_TXOUTCLKPCS_OUT),
    .TXRATEDONE_OUT              (GT1_TXRATEDONE_OUT),
    //------------ Transmit Ports - TX Initialization and Reset Ports ------------
    .TXRESETDONE_OUT             (GT1_TXRESETDONE_OUT),
    //---------------- Transmit Ports - TX Polarity Control Ports ----------------
    .TXPOLARITY_IN               (GT1_TXPOLARITY_IN),
    //---------------- Transmit Ports - pattern Generator Ports ----------------
    .TXPRBSSEL_IN                (GT1_TXPRBSSEL_IN),
    .TXPD_IN                     (GT1_TXPD_IN)

);

//_____________________________________________________________________________
//_____________________________________________________________________________
//GT2  (X0Y14)

gtx8_chan_pll_GT #
(
    // Simulation attributes
    .GT_SIM_GTRESET_SPEEDUP   (WRAPPER_SIM_GTRESET_SPEEDUP),
    .RX_DFE_KL_CFG2_IN        (RX_DFE_KL_CFG2_IN),
    .PCS_RSVD_ATTR_IN         (48'h000000000000),
    .PMA_RSV_IN               (PMA_RSV_IN)
)
gt2_gtx8_chan_pll_i
(
    //------------------------ Channel - Clocking Ports ------------------------
    .gtrefclk0_in                (GT2_GTREFCLK0_IN),
    //-------------------------- Channel - DRP Ports  --------------------------
    .DRPADDR_IN                  (GT2_DRPADDR_IN),
    .DRPCLK_IN                   (GT2_DRPCLK_IN),
    .DRPDI_IN                    (GT2_DRPDI_IN),
    .DRPDO_OUT                   (GT2_DRPDO_OUT),
    .DRPEN_IN                    (GT2_DRPEN_IN),
    .DRPRDY_OUT                  (GT2_DRPRDY_OUT),
    .DRPWE_IN                    (GT2_DRPWE_IN),
    //----------------------------- Clocking Ports -----------------------------
    .QPLLCLK_IN                  (gt2_qpllclk_i),
    .QPLLREFCLK_IN               (gt2_qpllrefclk_i),
    .RXSYSCLKSEL_IN              (GT2_RXSYSCLKSEL_IN),
    .TXSYSCLKSEL_IN              (GT2_TXSYSCLKSEL_IN),
    .RXOUTCLKSEL_IN              (GT2_RXOUTCLKSEL_IN),
    .TXOUTCLKSEL_IN              (GT2_TXOUTCLKSEL_IN),
    .RX8B10BEN_IN                (GT2_RX8B10BEN_IN),
    .TX8B10BEN_IN                (GT2_TX8B10BEN_IN),
    //---------------------------- Loopback Ports -----------------------------
    .LOOPBACK_IN                 (GT2_LOOPBACK_IN),
    //---------------------------- PCI Express Ports ----------------------------
    .RXRATE_IN                   (GT2_RXRATE_IN),
    //------------------- RX Initialization and Reset Ports --------------------
    .RXUSERRDY_IN                (GT2_RXUSERRDY_IN),
    //------------------------ RX Margin Analysis Ports ------------------------
    .EYESCANDATAERROR_OUT        (GT2_EYESCANDATAERROR_OUT),
    //----------------------- Receive Ports - CDR Ports -----------------------
```

```
    .RXCDRLOCK_OUT                (GT2_RXCDRLOCK_OUT),
    //---------- Receive Ports - FPGA RX Interface Ports -----------------
    .RXUSRCLK_IN                  (GT2_RXUSRCLK_IN),
    .RXUSRCLK2_IN                 (GT2_RXUSRCLK2_IN),
    //---------- Receive Ports - FPGA RX interface Ports -----------------
    .RXDATA_OUT                   (GT2_RXDATA_OUT),
    //---------- Receive Ports - Pattern Checker Ports -------------------
    .RXPRBSERR_OUT                (GT2_RXPRBSERR_OUT),
    .RXPRBSSEL_IN                 (GT2_RXPRBSSEL_IN),
    //---------- Receive Ports - Pattern Checker ports -------------------
    .RXPRBSCNTRESET_IN            (GT2_RXPRBSCNTRESET_IN),
    //------------------------ Receive Ports - RX AFE -------------------------
    .GTXRXP_IN                    (GT2_GTXRXP_IN),
    //---------------------- Receive Ports - RX AFE Ports ----------------------
    .GTXRXN_IN                    (GT2_GTXRXN_IN),
    //------------------- Receive Ports - RX Equalizer Ports -------------------
    .RXDFEAGCHOLD_IN              (GT2_RXDFEAGCHOLD_IN),
    .RXDFELFHOLD_IN               (GT2_RXDFELFHOLD_IN),
    //---------- Receive Ports - RX Fabric ClocK Output Control Ports ----------
    .RXRATEDONE_OUT               (GT2_RXRATEDONE_OUT),
    //------------- Receive Ports - RX Fabric Output Control Ports -------------
    .RXOUTCLK_OUT                 (GT2_RXOUTCLK_OUT),
    //----------- Receive Ports - RX Initialization and Reset Ports ------------
    .GTRXRESET_IN                 (GT2_GTRXRESET_IN),
    .RXPMARESET_IN                (GT2_RXPMARESET_IN),
    //---------------- Receive Ports - RX Polarity Control Ports ----------------
    .RXPOLARITY_IN                (GT2_RXPOLARITY_IN),
    //--------------------- Receive Ports - RX gearbox ports --------------------
    .RXSLIDE_IN                   (GT2_RXSLIDE_IN),
    //------------ Receive Ports -RX Initialization and Reset Ports ------------
    .RXRESETDONE_OUT              (GT2_RXRESETDONE_OUT),
    //---------------------- TX Configurable Driver Ports ----------------------
    .TXPOSTCURSOR_IN              (GT2_TXPOSTCURSOR_IN),
    .TXPRECURSOR_IN               (GT2_TXPRECURSOR_IN),
    //------------------- TX Initialization and Reset Ports --------------------
    .GTTXRESET_IN                 (GT2_GTTXRESET_IN),
    .TXUSERRDY_IN                 (GT2_TXUSERRDY_IN),
    //---------------- Transmit Ports - FPGA TX Interface Ports ----------------
    .TXUSRCLK_IN                  (GT2_TXUSRCLK_IN),
    .TXUSRCLK2_IN                 (GT2_TXUSRCLK2_IN),
    //------------------- Transmit Ports - PCI Express Ports -------------------
    .TXRATE_IN                    (GT2_TXRATE_IN),
    //---------------- Transmit Ports - Pattern Generator Ports ----------------
    .TXPRBSFORCEERR_IN            (GT2_TXPRBSFORCEERR_IN),
    //------------- Transmit Ports - TX Configurable Driver Ports --------------
    .TXDIFFCTRL_IN                (GT2_TXDIFFCTRL_IN),
    .TXMAINCURSOR_IN              (GT2_TXMAINCURSOR_IN),
    //---------------- Transmit Ports - TX Data Path interface -----------------
    .TXDATA_IN                    (GT2_TXDATA_IN),
    //-------------- Transmit Ports - TX Driver and OOB signaling --------------
    .GTXTXN_OUT                   (GT2_GTXTXN_OUT),
    .GTXTXP_OUT                   (GT2_GTXTXP_OUT),
    //--------- Transmit Ports - TX Fabric Clock Output Control Ports ----------
    .TXOUTCLK_OUT                 (GT2_TXOUTCLK_OUT),
    .TXOUTCLKFABRIC_OUT           (GT2_TXOUTCLKFABRIC_OUT),
    .TXOUTCLKPCS_OUT              (GT2_TXOUTCLKPCS_OUT),
    .TXRATEDONE_OUT               (GT2_TXRATEDONE_OUT),
    //----------- Transmit Ports - TX Initialization and Reset Ports -----------
    .TXRESETDONE_OUT              (GT2_TXRESETDONE_OUT),
    //---------------- Transmit Ports - TX Polarity Control Ports ----------------
    .TXPOLARITY_IN                (GT2_TXPOLARITY_IN),
    //---------------- Transmit Ports - pattern Generator Ports ----------------
    .TXPRBSSEL_IN                 (GT2_TXPRBSSEL_IN),
    .TXPD_IN                      (GT2_TXPD_IN)

);

//_____________________________________________________________________
//_____________________________________________________________________
//GT3  (X0Y15)
```

```
gtx8_chan_pll_GT #
(
   // Simulation attributes
   .GT_SIM_GTRESET_SPEEDUP   (WRAPPER_SIM_GTRESET_SPEEDUP),
   .RX_DFE_KL_CFG2_IN        (RX_DFE_KL_CFG2_IN),
   .PCS_RSVD_ATTR_IN         (48'h000000000000),
   .PMA_RSV_IN               (PMA_RSV_IN)
)
gt3_gtx8_chan_pll_i
(
   //------------------------ Channel - Clocking Ports ------------------------
   .gtrefclk0_in                (GT3_GTREFCLK0_IN),
   //-------------------------- Channel - DRP Ports  --------------------------
   .DRPADDR_IN                  (GT3_DRPADDR_IN),
   .DRPCLK_IN                   (GT3_DRPCLK_IN),
   .DRPDI_IN                    (GT3_DRPDI_IN),
   .DRPDO_OUT                   (GT3_DRPDO_OUT),
   .DRPEN_IN                    (GT3_DRPEN_IN),
   .DRPRDY_OUT                  (GT3_DRPRDY_OUT),
   .DRPWE_IN                    (GT3_DRPWE_IN),
   //----------------------------- Clocking Ports -----------------------------
   .QPLLCLK_IN                  (gt3_qpllclk_i),
   .QPLLREFCLK_IN               (gt3_qpllrefclk_i),
   .RXSYSCLKSEL_IN              (GT3_RXSYSCLKSEL_IN),
   .TXSYSCLKSEL_IN              (GT3_TXSYSCLKSEL_IN),
   .RXOUTCLKSEL_IN              (GT3_RXOUTCLKSEL_IN),
   .TXOUTCLKSEL_IN              (GT3_TXOUTCLKSEL_IN),
   .RX8B10BEN_IN                (GT3_RX8B10BEN_IN),
   .TX8B10BEN_IN                (GT3_TX8B10BEN_IN),
   //----------------------------- Loopback Ports -----------------------------
   .LOOPBACK_IN                 (GT3_LOOPBACK_IN),
   //--------------------------- PCI Express Ports ----------------------------
   .RXRATE_IN                   (GT3_RXRATE_IN),
   //------------------- RX Initialization and Reset Ports --------------------
   .RXUSERRDY_IN                (GT3_RXUSERRDY_IN),
   //------------------------ RX Margin Analysis Ports ------------------------
   .EYESCANDATAERROR_OUT        (GT3_EYESCANDATAERROR_OUT),
   //----------------------- Receive Ports - CDR Ports ------------------------
   .RXCDRLOCK_OUT               (GT3_RXCDRLOCK_OUT),
   //---------------- Receive Ports - FPGA RX Interface Ports -----------------
   .RXUSRCLK_IN                 (GT3_RXUSRCLK_IN),
   .RXUSRCLK2_IN                (GT3_RXUSRCLK2_IN),
   //---------------- Receive Ports - FPGA RX interface Ports -----------------
   .RXDATA_OUT                  (GT3_RXDATA_OUT),
   //----------------- Receive Ports - Pattern Checker Ports ------------------
   .RXPRBSERR_OUT               (GT3_RXPRBSERR_OUT),
   .RXPRBSSEL_IN                (GT3_RXPRBSSEL_IN),
   //----------------- Receive Ports - Pattern Checker ports ------------------
   .RXPRBSCNTRESET_IN           (GT3_RXPRBSCNTRESET_IN),
   //------------------------ Receive Ports - RX AFE --------------------------
   .GTXRXP_IN                   (GT3_GTXRXP_IN),
   //---------------------- Receive Ports - RX AFE Ports ----------------------
   .GTXRXN_IN                   (GT3_GTXRXN_IN),
   //------------------- Receive Ports - RX Equalizer Ports -------------------
   .RXDFEAGCHOLD_IN             (GT3_RXDFEAGCHOLD_IN),
   .RXDFELFHOLD_IN              (GT3_RXDFELFHOLD_IN),
   //---------- Receive Ports - RX Fabric ClocK Output Control Ports ----------
   .RXRATEDONE_OUT              (GT3_RXRATEDONE_OUT),
   //------------- Receive Ports - RX Fabric Output Control Ports -------------
   .RXOUTCLK_OUT                (GT3_RXOUTCLK_OUT),
   //----------- Receive Ports - RX Initialization and Reset Ports ------------
   .GTRXRESET_IN                (GT3_GTRXRESET_IN),
   .RXPMARESET_IN               (GT3_RXPMARESET_IN),
   //---------------- Receive Ports - RX Polarity Control Ports ---------------
   .RXPOLARITY_IN               (GT3_RXPOLARITY_IN),
   //-------------------- Receive Ports - RX gearbox ports --------------------
   .RXSLIDE_IN                  (GT3_RXSLIDE_IN),
   //------------ Receive Ports -RX Initialization and Reset Ports ------------
   .RXRESETDONE_OUT             (GT3_RXRESETDONE_OUT),
```

```
    //---------------------- TX Configurable Driver Ports ----------------------
    .TXPOSTCURSOR_IN              (GT3_TXPOSTCURSOR_IN),
    .TXPRECURSOR_IN               (GT3_TXPRECURSOR_IN),
    //------------------- TX Initialization and Reset Ports --------------------
    .GTTXRESET_IN                 (GT3_GTTXRESET_IN),
    .TXUSERRDY_IN                 (GT3_TXUSERRDY_IN),
    //---------------- Transmit Ports - FPGA TX Interface Ports ----------------
    .TXUSRCLK_IN                  (GT3_TXUSRCLK_IN),
    .TXUSRCLK2_IN                 (GT3_TXUSRCLK2_IN),
    //------------------ Transmit Ports - PCI Express Ports ------------------
    .TXRATE_IN                    (GT3_TXRATE_IN),
    //---------------- Transmit Ports - Pattern Generator Ports ----------------
    .TXPRBSFORCEERR_IN            (GT3_TXPRBSFORCEERR_IN),
    //------------- Transmit Ports - TX Configurable Driver Ports --------------
    .TXDIFFCTRL_IN                (GT3_TXDIFFCTRL_IN),
    .TXMAINCURSOR_IN              (GT3_TXMAINCURSOR_IN),
    //---------------- Transmit Ports - TX Data Path interface -----------------
    .TXDATA_IN                    (GT3_TXDATA_IN),
    //-------------- Transmit Ports - TX Driver and OOB signaling --------------
    .GTXTXN_OUT                   (GT3_GTXTXN_OUT),
    .GTXTXP_OUT                   (GT3_GTXTXP_OUT),
    //--------- Transmit Ports - TX Fabric Clock Output Control Ports ----------
    .TXOUTCLK_OUT                 (GT3_TXOUTCLK_OUT),
    .TXOUTCLKFABRIC_OUT           (GT3_TXOUTCLKFABRIC_OUT),
    .TXOUTCLKPCS_OUT              (GT3_TXOUTCLKPCS_OUT),
    .TXRATEDONE_OUT               (GT3_TXRATEDONE_OUT),
    //----------- Transmit Ports - TX Initialization and Reset Ports -----------
    .TXRESETDONE_OUT              (GT3_TXRESETDONE_OUT),
    //---------------- Transmit Ports - TX Polarity Control Ports ----------------
    .TXPOLARITY_IN                (GT3_TXPOLARITY_IN),
    //---------------- Transmit Ports - pattern Generator Ports ----------------
    .TXPRBSSEL_IN                 (GT3_TXPRBSSEL_IN),
    .TXPD_IN                      (GT3_TXPD_IN)

);



//_____________________________________________________________________________
//_____________________________________________________________________________
//_____________________________GTXE2_COMMON_____________________________________

GTXE2_COMMON #
(
      // Simulation attributes
      .SIM_RESET_SPEEDUP   (WRAPPER_SIM_GTRESET_SPEEDUP),
      .SIM_QPLLREFCLK_SEL  (3'b001),
      .SIM_VERSION        ("4.0"),


     //----------------COMMON BLOCK Attributes---------------
      .BIAS_CFG                       (64'h0000040000001000),
      .COMMON_CFG                     (32'h00000000),
      .QPLL_CFG                       (27'h06801E1),
      .QPLL_CLKOUT_CFG                (4'b0000),
      .QPLL_COARSE_FREQ_OVRD          (6'b010000),
      .QPLL_COARSE_FREQ_OVRD_EN       (1'b0),
      .QPLL_CP                        (10'b0000011111),
      .QPLL_CP_MONITOR_EN             (1'b0),
      .QPLL_DMONITOR_SEL              (1'b0),
      .QPLL_FBDIV                     (QPLL_FBDIV_IN),
      .QPLL_FBDIV_MONITOR_EN          (1'b0),
      .QPLL_FBDIV_RATIO               (QPLL_FBDIV_RATIO),
      .QPLL_INIT_CFG                  (24'h000006),
      .QPLL_LOCK_CFG                  (16'h21E8),
      .QPLL_LPF                       (4'b1111),
      .QPLL_REFCLK_DIV                (1)

)
gtxe2_common_0_i
```

```
    (
        //---------- Common Block  - Dynamic Reconfiguration Port (DRP) ----------
        .DRPADDR                    (GTC0_DRPADDR),
        .DRPCLK                     (GTC0_DRPCLK),
        .DRPDI                      (GTC0_DRPDI),
        .DRPDO                      (GTC0_DRPDO),
        .DRPEN                      (GTC0_DRPEN),
        .DRPRDY                     (GTC0_DRPRDY),
        .DRPWE                      (GTC0_DRPWE),
        //-------------------- Common Block  - Ref Clock Ports ---------------------
        .GTGREFCLK                  (tied_to_ground_i),
        .GTNORTHREFCLK0             (tied_to_ground_i),
        .GTNORTHREFCLK1             (tied_to_ground_i),
        .GTREFCLK0                  (GT0_GTREFCLK0_COMMON_IN),
        .GTREFCLK1                  (tied_to_ground_i),
        .GTSOUTHREFCLK0             (tied_to_ground_i),
        .GTSOUTHREFCLK1             (tied_to_ground_i),
        //----------------------- Common Block -  QPLL Ports ------------------------
        .QPLLDMONITOR               (),
        //---------------------- Common Block - Clocking Ports ----------------------
        .QPLLOUTCLK                 (gt0_qplloutclk_i),
        .QPLLOUTREFCLK              (gt0_qplloutrefclk_i),
        .REFCLKOUTMONITOR           (),
        //------------------------ Common Block - QPLL Ports ------------------------
        .QPLLFBCLKLOST              (),
        .QPLLLOCK                   (GT0_QPLLLOCK_OUT),
        .QPLLLOCKDETCLK             (GT0_QPLLLOCKDETCLK_IN),
        .QPLLLOCKEN                 (tied_to_vcc_i),
        .QPLLOUTRESET               (tied_to_ground_i),
        .QPLLPD                     (tied_to_vcc_i),
        .QPLLREFCLKLOST             (GT0_QPLLREFCLKLOST_OUT),
        .QPLLREFCLKSEL              (3'b001),
        .QPLLRESET                  (GT0_QPLLRESET_IN),
        .QPLLRSVD1                  (16'b0000000000000000),
        .QPLLRSVD2                  (5'b11111),
        //--------------------------------- QPLL Ports ---------------------------------
        .BGBYPASSB                  (tied_to_vcc_i),
        .BGMONITORENB               (tied_to_vcc_i),
        .BGPDB                      (tied_to_vcc_i),
        .BGRCALOVRD                 (5'b00000),
        .PMARSVD                    (8'b00000000),
        .RCALENB                    (tied_to_vcc_i)

    );

endmodule



module gtx8_chan_pll_GT #
(
    // Simulation attributes
    parameter   GT_SIM_GTRESET_SPEEDUP   =   "FALSE",       // Set to 1 to speed up sim reset;
    parameter   RX_DFE_KL_CFG2_IN        =   32'h301148AC,
    parameter   PMA_RSV_IN               =   32'h00018480,
    parameter   PCS_RSVD_ATTR_IN         =   48'h000000000000
)
(
    //------------------------- Channel - Clocking Ports -------------------------
    input          gtrefclk0_in,
    //--------------------------- Channel - DRP Ports  ---------------------------
    input   [8:0]   DRPADDR_IN,
    input          DRPCLK_IN,
    input   [15:0]  DRPDI_IN,
    output  [15:0]  DRPDO_OUT,
    input          DRPEN_IN,
    output          DRPRDY_OUT,
    input          DRPWE_IN,
    //----------------------------- Clocking Ports -----------------------------
```

```
input          QPLLCLK_IN,
input          QPLLREFCLK_IN,
input   [1:0]   RXSYSCLKSEL_IN,
input   [1:0]   TXSYSCLKSEL_IN,
input   [2:0]   RXOUTCLKSEL_IN,
input   [2:0]   TXOUTCLKSEL_IN,
input          RX8B10BEN_IN,
input          TX8B10BEN_IN,
//---------------------------- Loopback Ports -----------------------------
input   [2:0]   LOOPBACK_IN,
//---------------------------- PCI Express Ports ----------------------------
input   [2:0]   RXRATE_IN,
//------------------- RX Initialization and Reset Ports --------------------
input          RXUSERRDY_IN,
//------------------------ RX Margin Analysis Ports ------------------------
output          EYESCANDATAERROR_OUT,
//---------------------- Receive Ports - CDR Ports -----------------------
output          RXCDRLOCK_OUT,
//----------------- Receive Ports - FPGA RX Interface Ports -----------------
input          RXUSRCLK_IN,
input          RXUSRCLK2_IN,
//----------------- Receive Ports - FPGA RX interface Ports -----------------
output  [31:0]  RXDATA_OUT,
//----------------- Receive Ports - Pattern Checker Ports ------------------
output          RXPRBSERR_OUT,
input   [2:0]   RXPRBSSEL_IN,
//------------------ Receive Ports - Pattern Checker ports ------------------
input          RXPRBSCNTRESET_IN,
//------------------------ Receive Ports - RX AFE -------------------------
input          GTXRXP_IN,
//---------------------- Receive Ports - RX AFE Ports ----------------------
input          GTXRXN_IN,
//------------------- Receive Ports - RX Equalizer Ports -------------------
input          RXDFEAGCHOLD_IN,
input          RXDFELFHOLD_IN,
//---------- Receive Ports - RX Fabric ClocK Output Control Ports ----------
output          RXRATEDONE_OUT,
//------------- Receive Ports - RX Fabric Output Control Ports -------------
output          RXOUTCLK_OUT,
//----------- Receive Ports - RX Initialization and Reset Ports ------------
input          GTRXRESET_IN,
input          RXPMARESET_IN,
//---------------- Receive Ports - RX Polarity Control Ports ----------------
input          RXPOLARITY_IN,
//--------------------- Receive Ports - RX gearbox ports --------------------
input          RXSLIDE_IN,
//------------ Receive Ports -RX Initialization and Reset Ports ------------
output          RXRESETDONE_OUT,
//---------------------- TX Configurable Driver Ports ----------------------
input   [4:0]   TXPOSTCURSOR_IN,
input   [4:0]   TXPRECURSOR_IN,
//------------------- TX Initialization and Reset Ports --------------------
input          GTTXRESET_IN,
input          TXUSERRDY_IN,
//----------------- Transmit Ports - FPGA TX Interface Ports -----------------
input          TXUSRCLK_IN,
input          TXUSRCLK2_IN,
//------------------- Transmit Ports - PCI Express Ports -------------------
input   [2:0]   TXRATE_IN,
//----------------- Transmit Ports - Pattern Generator Ports -----------------
input          TXPRBSFORCEERR_IN,
//------------- Transmit Ports - TX Configurable Driver Ports --------------
input   [3:0]   TXDIFFCTRL_IN,
input   [6:0]   TXMAINCURSOR_IN,
//----------------- Transmit Ports - TX Data Path interface -----------------
input   [31:0]  TXDATA_IN,
//-------------- Transmit Ports - TX Driver and OOB signaling --------------
output          GTXTXN_OUT,
output          GTXTXP_OUT,
//--------- Transmit Ports - TX Fabric Clock Output Control Ports ----------
```

```
    output          TXOUTCLK_OUT,
    output          TXOUTCLKFABRIC_OUT,
    output          TXOUTCLKPCS_OUT,
    output          TXRATEDONE_OUT,
    //----------- Transmit Ports - TX Initialization and Reset Ports -----------
    output          TXRESETDONE_OUT,
    //---------------- Transmit Ports - TX Polarity Control Ports ----------------
    input           TXPOLARITY_IN,
    //---------------- Transmit Ports - pattern Generator Ports ----------------
    input   [2:0]   TXPRBSSEL_IN,
    //---------------- TX/RX powerdown control----------------- ----------------
    input   [1:0]   TXPD_IN


);


//***************************** Wire Declarations *****************************

    // ground and vcc signals
    wire            tied_to_ground_i;
    wire    [63:0]  tied_to_ground_vec_i;
    wire            tied_to_vcc_i;
    wire    [63:0]  tied_to_vcc_vec_i;


    //RX Datapath signals
    wire    [63:0]  rxdata_i;
    wire    [3:0]   rxchariscomma_float_i;
    wire    [3:0]   rxcharisk_float_i;
    wire    [3:0]   rxdisperr_float_i;
    wire    [3:0]   rxnotintable_float_i;
    wire    [3:0]   rxrundisp_float_i;


    //TX Datapath signals
    wire    [63:0]  txdata_i;
    wire    [3:0]   txkerr_float_i;
    wire    [3:0]   txrundisp_float_i;
    wire            rxstartofseq_float_i;
//
//*********************************** Main Body of Code**************************

    //-------------------------  Static signal Assigments ---------------------

    assign tied_to_ground_i             = 1'b0;
    assign tied_to_ground_vec_i         = 64'h0000000000000000;
    assign tied_to_vcc_i                = 1'b1;
    assign tied_to_vcc_vec_i            = 64'hffffffffffffffff;

    //-------------------  GT Datapath byte mapping  -----------------

    //The GT deserializes the rightmost parallel bit (LSb) first
    assign  RXDATA_OUT    =   rxdata_i[31:0];

    //The GT serializes the rightmost parallel bit (LSb) first
          assign txdata_i =   {tied_to_ground_vec_i[31:0], TXDATA_IN};




    //------------------------- GT Instantiations  -------------------------
       GTXE2_CHANNEL #
       (
          //_____________________________ Simulation-Only Attributes ____________________

          .SIM_RECEIVER_DETECT_PASS   ("TRUE"),
          .SIM_TX_EIDLE_DRIVE_LEVEL   ("X"),
          .SIM_RESET_SPEEDUP          (GT_SIM_GTRESET_SPEEDUP),
          .SIM_CPLLREFCLK_SEL         (3'b001),
```

```
 .SIM_VERSION              ("4.0"),


//----------------RX Byte and Word Alignment Attributes---------------
 .ALIGN_COMMA_DOUBLE                ("FALSE"),
 .ALIGN_COMMA_ENABLE                (10'b1111111111),
 .ALIGN_COMMA_WORD                  (1),
 .ALIGN_MCOMMA_DET                  ("TRUE"),
 .ALIGN_MCOMMA_VALUE                (10'b1010000011),
 .ALIGN_PCOMMA_DET                  ("TRUE"),
 .ALIGN_PCOMMA_VALUE                (10'b0101111100),
 .SHOW_REALIGN_COMMA                ("FALSE"),
 .RXSLIDE_AUTO_WAIT                 (7),
 .RXSLIDE_MODE                      ("PCS"),
 .RX_SIG_VALID_DLY                  (10),

//----------------RX 8B/10B Decoder Attributes----------------
 .RX_DISPERR_SEQ_MATCH              ("TRUE"),
 .DEC_MCOMMA_DETECT                 ("TRUE"),
 .DEC_PCOMMA_DETECT                 ("TRUE"),
 .DEC_VALID_COMMA_ONLY              ("FALSE"),

//-----------------------RX Clock Correction Attributes----------------------
 .CBCC_DATA_SOURCE_SEL              ("ENCODED"),
 .CLK_COR_SEQ_2_USE                 ("FALSE"),
 .CLK_COR_KEEP_IDLE                 ("FALSE"),
 .CLK_COR_MAX_LAT                   (19),
 .CLK_COR_MIN_LAT                   (15),
 .CLK_COR_PRECEDENCE                ("TRUE"),
 .CLK_COR_REPEAT_WAIT               (0),
 .CLK_COR_SEQ_LEN                   (1),
 .CLK_COR_SEQ_1_ENABLE              (4'b1111),
 .CLK_COR_SEQ_1_1                   (10'b0100000000),
 .CLK_COR_SEQ_1_2                   (10'b0000000000),
 .CLK_COR_SEQ_1_3                   (10'b0000000000),
 .CLK_COR_SEQ_1_4                   (10'b0000000000),
 .CLK_CORRECT_USE                   ("FALSE"),
 .CLK_COR_SEQ_2_ENABLE              (4'b1111),
 .CLK_COR_SEQ_2_1                   (10'b0100000000),
 .CLK_COR_SEQ_2_2                   (10'b0000000000),
 .CLK_COR_SEQ_2_3                   (10'b0000000000),
 .CLK_COR_SEQ_2_4                   (10'b0000000000),

//-----------------------RX Channel Bonding Attributes----------------------
 .CHAN_BOND_KEEP_ALIGN              ("FALSE"),
 .CHAN_BOND_MAX_SKEW                (1),
 .CHAN_BOND_SEQ_LEN                 (1),
 .CHAN_BOND_SEQ_1_1                 (10'b0000000000),
 .CHAN_BOND_SEQ_1_2                 (10'b0000000000),
 .CHAN_BOND_SEQ_1_3                 (10'b0000000000),
 .CHAN_BOND_SEQ_1_4                 (10'b0000000000),
 .CHAN_BOND_SEQ_1_ENABLE            (4'b1111),
 .CHAN_BOND_SEQ_2_1                 (10'b0000000000),
 .CHAN_BOND_SEQ_2_2                 (10'b0000000000),
 .CHAN_BOND_SEQ_2_3                 (10'b0000000000),
 .CHAN_BOND_SEQ_2_4                 (10'b0000000000),
 .CHAN_BOND_SEQ_2_ENABLE            (4'b1111),
 .CHAN_BOND_SEQ_2_USE               ("FALSE"),
 .FTS_DESKEW_SEQ_ENABLE             (4'b1111),
 .FTS_LANE_DESKEW_CFG               (4'b1111),
 .FTS_LANE_DESKEW_EN                ("FALSE"),

//-------------------------RX Margin Analysis Attributes----------------------------
 .ES_CONTROL                        (6'b000000),
 .ES_ERRDET_EN                      ("FALSE"),
 .ES_EYE_SCAN_EN                    ("TRUE"),
 .ES_HORZ_OFFSET                    (12'h000),
 .ES_PMA_CFG                        (10'b0000000000),
 .ES_PRESCALE                       (5'b00000),
 .ES_QUALIFIER                      (80'h00000000000000000000),
```

```
 .ES_QUAL_MASK                   (80'h00000000000000000000),
 .ES_SDATA_MASK                  (80'h00000000000000000000),
 .ES_VERT_OFFSET                 (9'b000000000),

//-----------------------FPGA RX Interface Attributes-------------------------
 .RX_DATA_WIDTH                  (32),

//-------------------------PMA Attributes----------------------------
 .OUTREFCLK_SEL_INV              (2'b11),
 .PMA_RSV                        (PMA_RSV_IN),
 .PMA_RSV2                       (16'h2090),
 .PMA_RSV3                       (2'b00),
 .PMA_RSV4                       (32'h00000000),
 .RX_BIAS_CFG                    (12'b000000000100),
 .DMONITOR_CFG                   (24'h000A00),
 .RX_CM_SEL                      (2'b01),
 .RX_CM_TRIM                     (3'b010),
 .RX_DEBUG_CFG                   (12'b000000000000),
 .RX_OS_CFG                      (13'b0000010000000),
 .TERM_RCAL_CFG                  (5'b10000),
 .TERM_RCAL_OVRD                 (1'b0),
 .TST_RSV                        (32'h00000000),
 .RX_CLK25_DIV                   (8),
 .TX_CLK25_DIV                   (8),
 .UCODEER_CLR                    (1'b0),

//-------------------------PCI Express Attributes----------------------------
 .PCS_PCIE_EN                    ("FALSE"),

//-------------------------PCS Attributes----------------------------
 .PCS_RSVD_ATTR                  (PCS_RSVD_ATTR_IN),

//-----------RX Buffer Attributes------------
 .RXBUF_ADDR_MODE                ("FAST"),
 .RXBUF_EIDLE_HI_CNT             (4'b1000),
 .RXBUF_EIDLE_LO_CNT             (4'b0000),
 .RXBUF_EN                       ("TRUE"),
 .RX_BUFFER_CFG                  (6'b000000),
 .RXBUF_RESET_ON_CB_CHANGE       ("TRUE"),
 .RXBUF_RESET_ON_COMMAALIGN      ("FALSE"),
 .RXBUF_RESET_ON_EIDLE           ("FALSE"),
 .RXBUF_RESET_ON_RATE_CHANGE     ("TRUE"),
 .RXBUFRESET_TIME                (5'b00001),
 .RXBUF_THRESH_OVFLW             (61),
 .RXBUF_THRESH_OVRD              ("FALSE"),
 .RXBUF_THRESH_UNDFLW            (4),
 .RXDLY_CFG                      (16'h001F),
 .RXDLY_LCFG                     (9'h030),
 .RXDLY_TAP_CFG                  (16'h0000),
 .RXPH_CFG                       (24'h000000),
 .RXPHDLY_CFG                    (24'h084020),
 .RXPH_MONITOR_SEL               (5'b00000),
 .RX_XCLK_SEL                    ("RXREC"),
 .RX_DDI_SEL                     (6'b000000),
 .RX_DEFER_RESET_BUF_EN          ("TRUE"),

//---------------------CDR Attributes-------------------------

//For GTX only: Display Port, HBR/RBR- set RXCDR_CFG=72'h0380008bff40200008

//For GTX only: Display Port, HBR2 -   set RXCDR_CFG=72'h038C008bff20200010
 .RXCDR_CFG                      (72'h03000023ff20400020),
 .RXCDR_FR_RESET_ON_EIDLE        (1'b0),
 .RXCDR_HOLD_DURING_EIDLE        (1'b0),
 .RXCDR_PH_RESET_ON_EIDLE        (1'b0),
 .RXCDR_LOCK_CFG                 (6'b010101),

//-----------------RX Initialization and Reset Attributes-------------------
 .RXCDRFREQRESET_TIME            (5'b00001),
 .RXCDRPHRESET_TIME              (5'b00001),
```

```
 .RXISCANRESET_TIME                  (5'b00001),
 .RXPCSRESET_TIME                   (5'b00001),
 .RXPMARESET_TIME                    (5'b00011),

//-----------------RX OOB Signaling Attributes-------------------
 .RXOOB_CFG                         (7'b0000110),

//-----------------------RX Gearbox Attributes---------------------------
 .RXGEARBOX_EN                      ("FALSE"),
 .GEARBOX_MODE                       (3'b000),

//-----------------------PRBS Detection Attribute-----------------------
 .RXPRBS_ERR_LOOPBACK                (1'b0),

//-----------Power-Down Attributes----------
 .PD_TRANS_TIME_FROM_P2              (12'h03c),
 .PD_TRANS_TIME_NONE_P2              (8'h3c),
 .PD_TRANS_TIME_TO_P2               (8'h64),

//-----------RX OOB Signaling Attributes----------
 .SAS_MAX_COM                       (64),
 .SAS_MIN_COM                      (36),
 .SATA_BURST_SEQ_LEN                 (4'b1111),
 .SATA_BURST_VAL                     (3'b100),
 .SATA_EIDLE_VAL                    (3'b100),
 .SATA_MAX_BURST                     (8),
 .SATA_MAX_INIT                     (21),
 .SATA_MAX_WAKE                      (7),
 .SATA_MIN_BURST                     (4),
 .SATA_MIN_INIT                     (12),
 .SATA_MIN_WAKE                      (4),

//-----------RX Fabric Clock Output Control Attributes----------
 .TRANS_TIME_RATE                    (8'h0E),

//------------TX Buffer Attributes----------------
 .TXBUF_EN                          ("TRUE"),
 .TXBUF_RESET_ON_RATE_CHANGE          ("TRUE"),
 .TXDLY_CFG                         (16'h001F),
 .TXDLY_LCFG                         (9'h030),
 .TXDLY_TAP_CFG                      (16'h0000),
 .TXPH_CFG                          (16'h0780),
 .TXPHDLY_CFG                        (24'h084020),
 .TXPH_MONITOR_SEL                   (5'b00000),
 .TX_XCLK_SEL                        ("TXOUT"),

//-----------------------FPGA TX Interface Attributes-------------------------
 .TX_DATA_WIDTH                      (32),

//-----------------------TX Configurable Driver Attributes--------------------------
 .TX_DEEMPH0                        (5'b00000),
 .TX_DEEMPH1                        (5'b00000),
 .TX_EIDLE_ASSERT_DELAY              (3'b110),
 .TX_EIDLE_DEASSERT_DELAY             (3'b100),
 .TX_LOOPBACK_DRIVE_HIZ              ("FALSE"),
 .TX_MAINCURSOR_SEL                  (1'b0),
 .TX_DRIVE_MODE                     ("DIRECT"),
 .TX_MARGIN_FULL_0                    (7'b1001110),
 .TX_MARGIN_FULL_1                    (7'b1001001),
 .TX_MARGIN_FULL_2                    (7'b1000101),
 .TX_MARGIN_FULL_3                    (7'b1000010),
 .TX_MARGIN_FULL_4                    (7'b1000000),
 .TX_MARGIN_LOW_0                     (7'b1000110),
 .TX_MARGIN_LOW_1                     (7'b1000100),
 .TX_MARGIN_LOW_2                     (7'b1000010),
 .TX_MARGIN_LOW_3                     (7'b1000000),
 .TX_MARGIN_LOW_4                     (7'b1000000),

//-----------------------TX Gearbox Attributes--------------------------
 .TXGEARBOX_EN                      ("FALSE"),
```

```
  //-----------------------TX Initialization and Reset Attributes--------------------------
   .TXPCSRESET_TIME                 (5'b00001),
   .TXPMARESET_TIME                 (5'b00001),

  //-----------------------TX Receiver Detection Attributes--------------------------
   .TX_RXDETECT_CFG                 (14'h1832),
   .TX_RXDETECT_REF                 (3'b100),

  //--------------------------CPLL Attributes----------------------------
   .CPLL_CFG                        (24'hBC07DC),
   .CPLL_FBDIV                       (5),
   .CPLL_FBDIV_45                    (5),
   .CPLL_INIT_CFG                    (24'h00001E),
   .CPLL_LOCK_CFG                    (16'h01E8),
   .CPLL_REFCLK_DIV                  (1),
   .RXOUT_DIV                       (1),
   .TXOUT_DIV                       (1),
   .SATA_CPLL_CFG                    ("VCO_3000MHZ"),

  //------------RX Initialization and Reset Attributes-------------
   .RXDFELPMRESET_TIME              (7'b0001111),

  //------------RX Equalizer Attributes-------------
   .RXLPM_HF_CFG                    (14'b00000011110000),
   .RXLPM_LF_CFG                    (14'b00000011110000),
   .RX_DFE_GAIN_CFG                 (23'h020FEA),
   .RX_DFE_H2_CFG                   (12'b000000000000),
   .RX_DFE_H3_CFG                   (12'b000001000000),
   .RX_DFE_H4_CFG                   (11'b00011110000),
   .RX_DFE_H5_CFG                   (11'b00011100000),
   .RX_DFE_KL_CFG                   (13'b0000011111110),
   .RX_DFE_LPM_CFG                  (16'h0954),
   .RX_DFE_LPM_HOLD_DURING_EIDLE    (1'b0),
   .RX_DFE_UT_CFG                   (17'b10001111000000000),
   .RX_DFE_VP_CFG                   (17'b00011111100000011),

  //-----------------------Power-Down Attributes----------------------------
   .RX_CLKMUX_PD                    (1'b1),
   .TX_CLKMUX_PD                    (1'b1),

  //-----------------------FPGA RX Interface Attribute-------------------------
   .RX_INT_DATAWIDTH                (1),

  //-----------------------FPGA TX Interface Attribute-------------------------
   .TX_INT_DATAWIDTH                (1),

  //----------------TX Configurable Driver Attributes---------------
   .TX_QPI_STATUS_EN                (1'b0),

  //-----------------------RX Equalizer Attributes--------------------------
   .RX_DFE_KL_CFG2                  (RX_DFE_KL_CFG2_IN),
   .RX_DFE_XYD_CFG                  (13'b0000000000000),

  //-----------------------TX Configurable Driver Attributes--------------------------
   .TX_PREDRIVER_MODE               (1'b0)


)
gtxe2_i
(

//------------------------------- CPLL Ports -------------------------------
.CPLLFBCLKLOST              (),
.CPLLLOCK                   (),
.CPLLLOCKDETCLK             (),
.CPLLLOCKEN                 (tied_to_vcc_i),
.CPLLPD                     (tied_to_ground_i),
.CPLLREFCLKLOST             (),
.CPLLREFCLKSEL              (3'b001),
```

```
.CPLLRESET                  (tied_to_ground_i),
.GTRSVD                     (16'b0000000000000000),
.PCSRSVDIN                  (16'b0000000000000000),
.PCSRSVDIN2                 (5'b00000),
.PMARSVDIN                  (5'b00000),
.PMARSVDIN2                 (5'b00000),
.TSTIN                      (20'b11111111111111111111),
.TSTOUT                     (),
//---------------------------------- Channel ----------------------------------
.CLKRSVD                    (4'b0000),
//------------------------ Channel - Clocking Ports ------------------------
.GTGREFCLK                  (tied_to_ground_i),
.GTNORTHREFCLK0             (tied_to_ground_i),
.GTNORTHREFCLK1             (tied_to_ground_i),
.GTREFCLK0                  (gtrefclk0_in),
.GTREFCLK1                  (tied_to_ground_i),
.GTSOUTHREFCLK0             (tied_to_ground_i),
.GTSOUTHREFCLK1             (tied_to_ground_i),
//-------------------------- Channel - DRP Ports  --------------------------
.DRPADDR                    (DRPADDR_IN),
.DRPCLK                     (DRPCLK_IN),
.DRPDI                      (DRPDI_IN),
.DRPDO                      (DRPDO_OUT),
.DRPEN                      (DRPEN_IN),
.DRPRDY                     (DRPRDY_OUT),
.DRPWE                      (DRPWE_IN),
//----------------------------- Clocking Ports -----------------------------
.GTREFCLKMONITOR            (),
.QPLLCLK                    (QPLLCLK_IN),
.QPLLREFCLK                 (QPLLREFCLK_IN),
.RXSYSCLKSEL                (2'b00),
.TXSYSCLKSEL                (2'b00),
//------------------------ Digital Monitor Ports --------------------------
.DMONITOROUT                (),
//---------- FPGA TX Interface Datapath Configuration  ----------
.TX8B10BEN                  (TX8B10BEN_IN),
//---------------------------- Loopback Ports -----------------------------
.LOOPBACK                   (LOOPBACK_IN),
//---------------------------- PCI Express Ports ----------------------------
.PHYSTATUS                  (),
.RXRATE                     (RXRATE_IN),
.RXVALID                    (),
//----------------------------- Power-Down Ports -----------------------------
.RXPD                       (2'b00),
.TXPD                       (TXPD_IN),
//------------------------ RX 8B/10B Decoder Ports -------------------------
.SETERRSTATUS               (tied_to_ground_i),
//------------------- RX Initialization and Reset Ports --------------------
.EYESCANRESET               (tied_to_ground_i),
.RXUSERRDY                  (RXUSERRDY_IN),
//------------------------ RX Margin Analysis Ports ------------------------
.EYESCANDATAERROR           (EYESCANDATAERROR_OUT),
.EYESCANMODE                (tied_to_ground_i),
.EYESCANTRIGGER             (tied_to_ground_i),
//----------------------- Receive Ports - CDR Ports ------------------------
.RXCDRFREQRESET             (tied_to_ground_i),
.RXCDRHOLD                  (tied_to_ground_i),
.RXCDRLOCK                  (RXCDRLOCK_OUT),
.RXCDROVRDEN                (tied_to_ground_i),
.RXCDRRESET                 (tied_to_ground_i),
.RXCDRRESETRSV              (tied_to_ground_i),
//----------------- Receive Ports - Clock Correction Ports -----------------
.RXCLKCORCNT                (),
//-------- Receive Ports - FPGA RX Interface Datapath Configuration --------
.RX8B10BEN                  (RX8B10BEN_IN),
//---------------- Receive Ports - FPGA RX Interface Ports -----------------
.RXUSRCLK                   (RXUSRCLK_IN),
.RXUSRCLK2                  (RXUSRCLK2_IN),
//---------------- Receive Ports - FPGA RX interface Ports -----------------
.RXDATA                     (rxdata_i),
```

```
//---------------- Receive Ports - Pattern Checker Ports ------------------
.RXPRBSERR                  (RXPRBSERR_OUT),
.RXPRBSSEL                  (RXPRBSSEL_IN),
//---------------- Receive Ports - Pattern Checker ports ------------------
.RXPRBSCNTRESET             (RXPRBSCNTRESET_IN),
//------------------ Receive Ports - RX  Equalizer Ports --------------------
.RXDFEXYDEN                 (tied_to_vcc_i),
.RXDFEXYDHOLD               (tied_to_ground_i),
.RXDFEXYDOVRDEN             (tied_to_ground_i),
//---------------- Receive Ports - RX 8B/10B Decoder Ports -----------------
.RXDISPERR                  (),
.RXNOTINTABLE               (),
//------------------------ Receive Ports - RX AFE -------------------------
.GTXRXP                     (GTXRXP_IN),
//---------------------- Receive Ports - RX AFE Ports ----------------------
.GTXRXN                     (GTXRXN_IN),
//---------------- Receive Ports - RX Buffer Bypass Ports -----------------
.RXBUFRESET                 (tied_to_ground_i),
.RXBUFSTATUS                (),
.RXDDIEN                    (tied_to_ground_i),
.RXDLYBYPASS                (tied_to_vcc_i),
.RXDLYEN                    (tied_to_ground_i),
.RXDLYOVRDEN                (tied_to_ground_i),
.RXDLYSRESET                (tied_to_ground_i),
.RXDLYSRESETDONE            (),
.RXPHALIGN                  (tied_to_ground_i),
.RXPHALIGNDONE              (),
.RXPHALIGNEN                (tied_to_ground_i),
.RXPHDLYPD                  (tied_to_ground_i),
.RXPHDLYRESET               (tied_to_ground_i),
.RXPHMONITOR                (),
.RXPHOVRDEN                 (tied_to_ground_i),
.RXPHSLIPMONITOR            (),
.RXSTATUS                   (),
//------------ Receive Ports - RX Byte and Word Alignment Ports ------------
.RXBYTEISALIGNED            (),
.RXBYTEREALIGN              (),
.RXCOMMADET                 (),
.RXCOMMADETEN               (tied_to_vcc_i),
.RXMCOMMAALIGNEN            (tied_to_ground_i),
.RXPCOMMAALIGNEN            (tied_to_ground_i),
//---------------- Receive Ports - RX Channel Bonding Ports ----------------
.RXCHANBONDSEQ              (),
.RXCHBONDEN                 (tied_to_ground_i),
.RXCHBONDLEVEL              (tied_to_ground_vec_i[2:0]),
.RXCHBONDMASTER             (tied_to_ground_i),
.RXCHBONDO                  (),
.RXCHBONDSLAVE              (tied_to_ground_i),
//---------------- Receive Ports - RX Channel Bonding Ports  ----------------
.RXCHANISALIGNED            (),
.RXCHANREALIGN              (),
//------------------ Receive Ports - RX Equailizer Ports --------------------
.RXLPMHFHOLD                (tied_to_ground_i),
.RXLPMHFOVRDEN              (tied_to_ground_i),
.RXLPMLFHOLD                (tied_to_ground_i),
//------------------- Receive Ports - RX Equalizer Ports --------------------
.RXDFEAGCHOLD               (RXDFEAGCHOLD_IN),
.RXDFEAGCOVRDEN             (tied_to_ground_i),
.RXDFECM1EN                 (tied_to_ground_i),
.RXDFELFHOLD                (RXDFELFHOLD_IN),
.RXDFELFOVRDEN              (tied_to_vcc_i),
.RXDFELPMRESET              (tied_to_ground_i),
.RXDFETAP2HOLD              (tied_to_ground_i),
.RXDFETAP2OVRDEN            (tied_to_ground_i),
.RXDFETAP3HOLD              (tied_to_ground_i),
.RXDFETAP3OVRDEN            (tied_to_ground_i),
.RXDFETAP4HOLD              (tied_to_ground_i),
.RXDFETAP4OVRDEN            (tied_to_ground_i),
.RXDFETAP5HOLD              (tied_to_ground_i),
.RXDFETAP5OVRDEN            (tied_to_ground_i),
```

```
.RXDFEUTHOLD                    (tied_to_ground_i),
.RXDFEUTOVRDEN                  (tied_to_ground_i),
.RXDFEVPHOLD                    (tied_to_ground_i),
.RXDFEVPOVRDEN                  (tied_to_ground_i),
.RXDFEVSEN                      (tied_to_ground_i),
.RXLPMLFKLOVRDEN                (tied_to_ground_i),
.RXMONITOROUT                   (),
.RXMONITORSEL                   (2'b00),
.RXOSHOLD                       (tied_to_ground_i),
.RXOSOVRDEN                     (tied_to_ground_i),
//---------- Receive Ports - RX Fabric ClocK Output Control Ports ----------
.RXRATEDONE                     (RXRATEDONE_OUT),
//------------- Receive Ports - RX Fabric Output Control Ports -------------
.RXOUTCLK                       (RXOUTCLK_OUT),
.RXOUTCLKFABRIC                 (),
.RXOUTCLKPCS                    (),
.RXOUTCLKSEL                    (RXOUTCLKSEL_IN),
//------------------- Receive Ports - RX Gearbox Ports --------------------
.RXDATAVALID                    (),
.RXHEADER                       (),
.RXHEADERVALID                  (),
.RXSTARTOFSEQ                   (),
//------------------- Receive Ports - RX Gearbox Ports  --------------------
.RXGEARBOXSLIP                  (tied_to_ground_i),
//----------- Receive Ports - RX Initialization and Reset Ports ------------
.GTRXRESET                      (GTRXRESET_IN),
.RXOOBRESET                     (tied_to_ground_i),
.RXPCSRESET                     (tied_to_ground_i),
.RXPMARESET                     (RXPMARESET_IN),
//---------------- Receive Ports - RX Margin Analysis ports ----------------
.RXLPMEN                        (1'b1),
//----------------- Receive Ports - RX OOB Signaling ports -----------------
.RXCOMSASDET                    (),
.RXCOMWAKEDET                   (),
//---------------- Receive Ports - RX OOB Signaling ports  -----------------
.RXCOMINITDET                   (),
//---------------- Receive Ports - RX OOB signalling Ports -----------------
.RXELECIDLE                     (),
.RXELECIDLEMODE                 (2'b11),
//--------------- Receive Ports - RX Polarity Control Ports ----------------
.RXPOLARITY                     (RXPOLARITY_IN),
//------------------- Receive Ports - RX gearbox ports --------------------
.RXSLIDE                        (RXSLIDE_IN),
//----------------- Receive Ports - RX8B/10B Decoder Ports -----------------
.RXCHARISCOMMA                  (),
.RXCHARISK                      (),
//---------------- Receive Ports - Rx Channel Bonding Ports ----------------
.RXCHBONDI                      (5'b00000),
//------------ Receive Ports -RX Initialization and Reset Ports ------------
.RXRESETDONE                    (RXRESETDONE_OUT),
//------------------------------ Rx AFE Ports ------------------------------
.RXQPIEN                        (tied_to_ground_i),
.RXQPISENN                      (),
.RXQPISENP                      (),
//------------------------ TX Buffer Bypass Ports ------------------------
.TXPHDLYTSTCLK                  (tied_to_ground_i),
//--------------------- TX Configurable Driver Ports ---------------------
.TXPOSTCURSOR                   (5'b1010),
.TXPOSTCURSORINV                (tied_to_ground_i),
.TXPRECURSOR                    (TXPRECURSOR_IN),
.TXPRECURSORINV                 (tied_to_ground_i),
.TXQPIBIASEN                    (tied_to_ground_i),
.TXQPISTRONGPDOWN               (tied_to_ground_i),
.TXQPIWEAKPUP                   (tied_to_ground_i),
//------------------- TX Initialization and Reset Ports --------------------
.CFGRESET                       (tied_to_ground_i),
.GTTXRESET                      (GTTXRESET_IN),
.PCSRSVDOUT                     (),
.TXUSERRDY                      (TXUSERRDY_IN),
//--------------------- Transceiver Reset Mode Operation --------------------
```

```
.GTRESETSEL                 (tied_to_ground_i),
.RESETOVRD                  (tied_to_ground_i),
//--------------- Transmit Ports - 8b10b Encoder Control Ports ---------------
.TXCHARDISPMODE             (tied_to_ground_vec_i[7:0]),
.TXCHARDISPVAL              (tied_to_ground_vec_i[7:0]),
//---------------- Transmit Ports - FPGA TX Interface Ports ----------------
.TXUSRCLK                   (TXUSRCLK_IN),
.TXUSRCLK2                  (TXUSRCLK2_IN),
//------------------- Transmit Ports - PCI Express Ports -------------------
.TXELECIDLE                 (tied_to_ground_i),
.TXMARGIN                   (tied_to_ground_vec_i[2:0]),
.TXRATE                     (TXRATE_IN),
.TXSWING                    (tied_to_ground_i),
//---------------- Transmit Ports - Pattern Generator Ports ----------------
.TXPRBSFORCEERR             (TXPRBSFORCEERR_IN),
//---------------- Transmit Ports - TX Buffer Bypass Ports -----------------
.TXDLYBYPASS                (tied_to_vcc_i),
.TXDLYEN                    (tied_to_ground_i),
.TXDLYHOLD                  (tied_to_ground_i),
.TXDLYOVRDEN                (tied_to_ground_i),
.TXDLYSRESET                (tied_to_ground_i),
.TXDLYSRESETDONE            (),
.TXDLYUPDOWN                (tied_to_ground_i),
.TXPHALIGN                  (tied_to_ground_i),
.TXPHALIGNDONE              (),
.TXPHALIGNEN                (tied_to_ground_i),
.TXPHDLYPD                  (tied_to_ground_i),
.TXPHDLYRESET               (tied_to_ground_i),
.TXPHINIT                   (tied_to_ground_i),
.TXPHINITDONE               (),
.TXPHOVRDEN                 (tied_to_ground_i),
//--------------------- Transmit Ports - TX Buffer Ports ---------------------
.TXBUFSTATUS                (),
//------------- Transmit Ports - TX Configurable Driver Ports --------------
.TXBUFDIFFCTRL              (3'b100),
.TXDEEMPH                   (tied_to_ground_i),
.TXDIFFCTRL                 (4'b1111),
.TXDIFFPD                   (tied_to_ground_i),
.TXINHIBIT                  (tied_to_ground_i),
.TXMAINCURSOR               (TXMAINCURSOR_IN),
.TXPISOPD                   (tied_to_ground_i),
//---------------- Transmit Ports - TX Data Path interface -----------------
.TXDATA                     (txdata_i),
//-------------- Transmit Ports - TX Driver and OOB signaling --------------
.GTXTXN                     (GTXTXN_OUT),
.GTXTXP                     (GTXTXP_OUT),
//--------- Transmit Ports - TX Fabric Clock Output Control Ports ----------
.TXOUTCLK                   (TXOUTCLK_OUT),
.TXOUTCLKFABRIC             (TXOUTCLKFABRIC_OUT),
.TXOUTCLKPCS                (TXOUTCLKPCS_OUT),
.TXOUTCLKSEL                (TXOUTCLKSEL_IN),
.TXRATEDONE                 (TXRATEDONE_OUT),
//------------------- Transmit Ports - TX Gearbox Ports --------------------
.TXCHARISK                  (tied_to_ground_vec_i[7:0]),
.TXGEARBOXREADY             (),
.TXHEADER                   (tied_to_ground_vec_i[2:0]),
.TXSEQUENCE                 (tied_to_ground_vec_i[6:0]),
.TXSTARTSEQ                 (tied_to_ground_i),
//----------- Transmit Ports - TX Initialization and Reset Ports -----------
.TXPCSRESET                 (tied_to_ground_i),
.TXPMARESET                 (tied_to_ground_i),
.TXRESETDONE                (TXRESETDONE_OUT),
//---------------- Transmit Ports - TX OOB signalling Ports ----------------
.TXCOMFINISH                (),
.TXCOMINIT                  (tied_to_ground_i),
.TXCOMSAS                   (tied_to_ground_i),
.TXCOMWAKE                  (tied_to_ground_i),
.TXPDELECIDLEMODE           (tied_to_ground_i),
//--------------- Transmit Ports - TX Polarity Control Ports ---------------
.TXPOLARITY                 (TXPOLARITY_IN),
```

```
        //------------ Transmit Ports - TX Receiver Detection Ports  --------------
        .TXDETECTRX                 (tied_to_ground_i),
        //---------------- Transmit Ports - TX8b/10b Encoder Ports -----------------
        .TX8B10BBYPASS              (tied_to_vcc_vec_i[7:0]),
        //---------------- Transmit Ports - pattern Generator Ports ----------------
        .TXPRBSSEL                  (TXPRBSSEL_IN),
        //---------------------- Tx Configurable Driver  Ports ----------------------
        .TXQPISENN                  (),
        .TXQPISENP                  ()

      );

  endmodule



  module gtx8_chan_pll_TX_STARTUP_FSM  #
    (
      parameter    GT_TYPE               = "GTX",
      parameter    STABLE_CLOCK_PERIOD     = 8,       // Period of the stable clock driving this state-machine, unit is [ns]
      parameter    RETRY_COUNTER_BITWIDTH   = 8,
      parameter    TX_QPLL_USED            = "FALSE",  // the TX and RX Reset FSMs must
      parameter    RX_QPLL_USED            = "FALSE",  // share these two generic values
      parameter     PHASE_ALIGNMENT_MANUAL    = "TRUE"     // Decision if a manual phase-alignment is necessary or the
automatic
                                        // is enough. For single-lane applications the automatic alignment is
                                        // sufficient
    )
    (
      input  wire     STABLE_CLOCK,            //Stable Clock, either a stable clock from the PCB
      input  wire     TXUSERCLK,              //TXUSERCLK as used in the design
      input  wire     SOFT_RESET,             //User Reset, can be pulled any time
      input  wire     QPLLREFCLKLOST,          //QPLL Reference-clock for the GT is lost
      input  wire     CPLLREFCLKLOST,          //CPLL Reference-clock for the GT is lost
      input  wire     QPLLLOCK,               //Lock Detect from the QPLL of the GT
      input  wire     CPLLLOCK ,              //Lock Detect from the CPLL of the GT
      input  wire     TXRESETDONE,
      input  wire     MMCM_LOCK,
      output reg      GTTXRESET  = 1'b0,
      output reg      MMCM_RESET = 1'b1,
      output reg      QPLL_RESET = 1'b0,       //Reset QPLL
      output reg      CPLL_RESET = 1'b0,       //Reset CPLL
      output          TX_FSM_RESET_DONE,       //Reset-sequence has sucessfully been finished.
      output reg      TXUSERRDY  = 1'b0,
      output          RUN_PHALIGNMENT,
      output reg      RESET_PHALIGNMENT = 1'b0,
      input  wire     PHALIGNMENT_DONE,

      output [RETRY_COUNTER_BITWIDTH-1:0] RETRY_COUNTER // Number of
                                        // Retries it took to get the transceiver up and running
    );


  //Interdependencies:
  // * Timing depends on the frequency of the stable clock. Hence counters-sizes
  //   are calculated at design-time based on the Generics
  //
  // * if either of the PLLs is reset during TX-startup, it does not need to be reset again by RX
  //   => signal which PLL has been reset
  // *


    localparam [2:0]
              INIT = 3'b000,
              ASSERT_ALL_RESETS = 3'b001,
              RELEASE_PLL_RESET = 3'b010,
              RELEASE_MMCM_RESET = 3'b011,
              WAIT_RESET_DONE = 3'b100,
              DO_PHASE_ALIGNMENT = 3'b101,
              RESET_FSM_DONE = 3'b110;
```

```
    reg [2:0] tx_state = INIT;


    parameter integer MMCM_LOCK_CNT_MAX = 1024;
    parameter integer STARTUP_DELAY    = 500;//AR43482: Transceiver needs to wait for 500 ns after configuration
    parameter integer WAIT_CYCLES      = STARTUP_DELAY / STABLE_CLOCK_PERIOD; // Number of Clock-Cycles to wait
after configuration
    parameter integer WAIT_MAX         = WAIT_CYCLES + 10;                 // 500 ns plus some additional margin

    parameter integer WAIT_TIMEOUT_2ms   = 2000000 / STABLE_CLOCK_PERIOD;//  2 ms time-out
    parameter integer WAIT_TLOCK_MAX     =  100000 / STABLE_CLOCK_PERIOD;//100 us time-out
    parameter integer WAIT_TIMEOUT_500us =  500000 / STABLE_CLOCK_PERIOD;//100 us time-out

    reg [7:0] init_wait_count = 0;
    reg       init_wait_done = 1'b0;
    reg       pll_reset_asserted = 1'b0;

    reg       tx_fsm_reset_done_int    = 1'b0;
    wire      tx_fsm_reset_done_int_s2;
    reg       tx_fsm_reset_done_int_s3 = 1'b0;

    parameter integer MAX_RETRIES         = 2**RETRY_COUNTER_BITWIDTH-1;
    reg [7:0] retry_counter_int = 0;
    reg [18:0] time_out_counter = 0;

    reg      reset_time_out = 1'b0;
    reg      time_out_2ms   = 1'b0;  //--\Flags that the various time-out points
    reg      time_tlock_max = 1'b0;  //--|have been reached.
    reg      time_out_500us = 1'b0;  //--/

    reg [9:0] mmcm_lock_count = 0;
    reg       mmcm_lock_int = 1'b0;
    wire      mmcm_lock_i;
    reg       mmcm_lock_reclocked = 1'b0;

    reg       run_phase_alignment_int = 1'b0;
    wire      run_phase_alignment_int_s2;
    reg       run_phase_alignment_int_s3 = 1'b0;
    parameter integer MAX_WAIT_BYPASS      = 75264;


    reg [16:0] wait_bypass_count = 0;
    reg       time_out_wait_bypass    = 1'b0;
    wire      time_out_wait_bypass_s2;
    reg       time_out_wait_bypass_s3 = 1'b0;

    wire      txresetdone_s2;
    reg       txresetdone_s3 = 1'b0;

    wire      refclk_lost;

    wire      cplllock_sync;
    wire      qpllock_sync;
    reg       cplllock_ris_edge = 1'b0;
    reg       qpllock_ris_edge = 1'b0;
    reg       cplllock_prev;
    reg       qpllock_prev;

    //Alias section, signals used within this module mapped to output ports:
   assign RETRY_COUNTER     = retry_counter_int;
   assign RUN_PHALIGNMENT   = run_phase_alignment_int;
   assign TX_FSM_RESET_DONE = tx_fsm_reset_done_int;


    always @(posedge STABLE_CLOCK)
    begin
       // The counter starts running when configuration has finished and
       // the clock is stable. When its maximum count-value has been reached,
       // the 500 ns from Answer Record 43482 have been passed.
```

```
    if (init_wait_count == WAIT_MAX)
       init_wait_done <= `DLY  1'b1;
    else
      init_wait_count <= `DLY  init_wait_count + 1;
 end


 always @(posedge STABLE_CLOCK)
 begin
    // One common large counter for generating three time-out signals.
    // Intermediate time-outs are derived from calculated values, based
    // on the period of the provided clock.
    if (reset_time_out == 1'b1)
    begin
      time_out_counter  <= `DLY  0;
      time_out_2ms      <= `DLY  1'b0;
      time_tlock_max    <= `DLY  1'b0;
      time_out_500us    <= `DLY  1'b0;
    end
    else
    begin
      if (time_out_counter == WAIT_TIMEOUT_2ms)
        time_out_2ms <= `DLY  1'b1;
      else
        time_out_counter <= `DLY  time_out_counter + 1;

      if (time_out_counter == WAIT_TLOCK_MAX)
        time_tlock_max <= `DLY  1'b1;

      if (time_out_counter == WAIT_TIMEOUT_500us)
        time_out_500us <= `DLY  1'b1;
    end
 end

 always @(posedge STABLE_CLOCK)
 begin
    if (mmcm_lock_i == 1'b0)
    begin
      mmcm_lock_count <= `DLY  0;
      mmcm_lock_reclocked   <= `DLY  1'b0;
    end
    else
    begin
      if (mmcm_lock_count < MMCM_LOCK_CNT_MAX - 1)
        mmcm_lock_count <= `DLY  mmcm_lock_count + 1;
      else
        mmcm_lock_reclocked <= `DLY  1'b1;
    end
 end


 //Clock Domain Crossing
gtx8_chan_pll_sync_block sync_run_phase_alignment_int
      (
        .clk            (TXUSERCLK),
        .data_in        (run_phase_alignment_int),
        .data_out       (run_phase_alignment_int_s2)
      );

gtx8_chan_pll_sync_block sync_tx_fsm_reset_done_int
      (
        .clk            (TXUSERCLK),
        .data_in        (tx_fsm_reset_done_int),
        .data_out       (tx_fsm_reset_done_int_s2)
      );

 always @(posedge TXUSERCLK)
 begin
   run_phase_alignment_int_s3 <= `DLY run_phase_alignment_int_s2;
   tx_fsm_reset_done_int_s3   <= `DLY tx_fsm_reset_done_int_s2;
```

```
 end

gtx8_chan_pll_sync_block sync_time_out_wait_bypass
      (
        .clk            (STABLE_CLOCK),
        .data_in        (time_out_wait_bypass),
        .data_out       (time_out_wait_bypass_s2)
      );

 gtx8_chan_pll_sync_block sync_TXRESETDONE
      (
        .clk            (STABLE_CLOCK),
        .data_in        (TXRESETDONE),
        .data_out       (txresetdone_s2)
      );

 gtx8_chan_pll_sync_block sync_mmcm_lock_reclocked
      (
        .clk            (STABLE_CLOCK),
        .data_in        (MMCM_LOCK),
        .data_out       (mmcm_lock_i)
      );

 gtx8_chan_pll_sync_block sync_cplllock
      (
        .clk            (STABLE_CLOCK),
        .data_in        (CPLLLOCK),
        .data_out       (cplllock_sync)
      );

 gtx8_chan_pll_sync_block sync_qplllock
      (
        .clk            (STABLE_CLOCK),
        .data_in        (QPLLLOCK),
        .data_out       (qplllock_sync)
      );


 always @(posedge STABLE_CLOCK)
 begin
   time_out_wait_bypass_s3   <= `DLY time_out_wait_bypass_s2;
   txresetdone_s3            <= `DLY txresetdone_s2;
   cplllock_prev             <= `DLY cplllock_sync;
   qplllock_prev             <= `DLY qplllock_sync;
 end

 always @(posedge STABLE_CLOCK)
 begin
   if(SOFT_RESET == 1'b1)
     cplllock_ris_edge <= 1'b0;
   else if((cplllock_prev == 1'b0) && (cplllock_sync == 1'b1))
     cplllock_ris_edge <= 1'b1;
   else if(tx_state == ASSERT_ALL_RESETS || tx_state == RELEASE_PLL_RESET)
     cplllock_ris_edge <= cplllock_ris_edge;
   else
     cplllock_ris_edge <= 1'b0;
 end

 always @(posedge STABLE_CLOCK)
 begin
   if((qplllock_prev == 1'b0) && (qplllock_sync == 1'b1))
     qplllock_ris_edge <= 1'b1;
   else if(tx_state == ASSERT_ALL_RESETS || tx_state == RELEASE_PLL_RESET)
     qplllock_ris_edge <= qplllock_ris_edge;
   else
     qplllock_ris_edge <= 1'b0;
 end


 always @(posedge TXUSERCLK)
```

```
begin
   if (run_phase_alignment_int_s3 == 1'b0)
   begin
     wait_bypass_count     <= `DLY  0;
     time_out_wait_bypass  <= `DLY  1'b0;
   end
   else if (run_phase_alignment_int_s3 == 1'b1 && tx_fsm_reset_done_int_s3 == 1'b0)
   begin
     if (wait_bypass_count == MAX_WAIT_BYPASS - 1)
       time_out_wait_bypass <= `DLY  1'b1;
     else
       wait_bypass_count <= `DLY  wait_bypass_count + 1;
   end
end

assign refclk_lost = ( TX_QPLL_USED == "TRUE"  && QPLLREFCLKLOST == 1'b1) ? 1'b1 :
                ( TX_QPLL_USED == "FALSE" && CPLLREFCLKLOST == 1'b1) ? 1'b1 : 1'b0;



//FSM for resetting the GTX/GTH/GTP in the 7-series.
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//
// Following steps are performed:
// 1) Only for GTX - After configuration wait for approximately 500 ns as specified in
//    answer-record 43482
// 2) Assert all resets on the GT and on an MMCM potentially connected.
//    After that wait until a reference-clock has been detected.
// 3) Release the reset to the GT and wait until the GT-PLL has locked.
// 4) Release the MMCM-reset and wait until the MMCM has signalled lock.
//    Also signal to the RX-side which PLL has been reset.
// 5) Wait for the RESET_DONE-signal from the GTX.
// 6) Signal to start the phase-alignment procedure and wait for it to
//    finish.
// 7) Reset-sequence has successfully run through. Signal this to the
//    rest of the design by asserting TX_FSM_RESET_DONE.

always @(posedge STABLE_CLOCK)
begin
   if (SOFT_RESET == 1'b1 || (tx_state != INIT && tx_state != ASSERT_ALL_RESETS && refclk_lost == 1'b1))
   begin
     tx_state              <= `DLY  INIT;
     TXUSERRDY             <= `DLY  1'b0;
     GTTXRESET             <= `DLY  1'b0;
     MMCM_RESET            <= `DLY  1'b1;
     tx_fsm_reset_done_int  <= `DLY  1'b0;
     QPLL_RESET            <= `DLY  1'b0;
     CPLL_RESET            <= `DLY  1'b0;
     pll_reset_asserted     <= `DLY  1'b0;
     reset_time_out        <= `DLY  1'b0;
     retry_counter_int      <= `DLY   0;
     run_phase_alignment_int <= `DLY  1'b0;
     RESET_PHALIGNMENT      <= `DLY  1'b1;
   end
   else
   begin
     case (tx_state)
       INIT :
       begin
        //Initial state after configuration. This state will be left after
        //approx. 500 ns and not be re-entered.
        if (init_wait_done == 1'b1)
          tx_state          <= `DLY  ASSERT_ALL_RESETS;
          reset_time_out    <= `DLY  1'b1;
       end

       ASSERT_ALL_RESETS :
       begin
        //This is the state into which the FSM will always jump back if any
        //time-outs will occur.
```

```
 //The number of retries is reported on the output RETRY_COUNTER. In
 //case the transceiver never comes up for some reason, this machine
 //will still continue its best and rerun until the FPGA is turned off
 //or the transceivers come up correctly.
 if (TX_QPLL_USED == "TRUE")
 begin
  if (pll_reset_asserted == 1'b0)
  begin
    QPLL_RESET         <= `DLY  1'b1;
    pll_reset_asserted  <= `DLY  1'b1;
  end
  else
    QPLL_RESET         <= `DLY  1'b0;
 end
 else
 begin
  if (pll_reset_asserted == 1'b0)
  begin
    CPLL_RESET <= `DLY  1'b1;
    pll_reset_asserted  <= `DLY  1'b1;
  end
  else
    CPLL_RESET         <= `DLY  1'b0;
 end
 TXUSERRDY             <= `DLY  1'b0;
 GTTXRESET             <= `DLY  1'b1;
 MMCM_RESET              <= `DLY  1'b1;
 reset_time_out        <= `DLY  1'b0;
 run_phase_alignment_int <= `DLY  1'b0;
 RESET_PHALIGNMENT       <= `DLY  1'b1;

 if ((TX_QPLL_USED == "TRUE" && QPLLREFCLKLOST == 1'b0 && pll_reset_asserted) ||
   (TX_QPLL_USED == "FALSE" && CPLLREFCLKLOST == 1'b0 && pll_reset_asserted))
  tx_state  <= `DLY  RELEASE_PLL_RESET;

end

RELEASE_PLL_RESET :
begin
 //PLL-Reset of the GTX gets released and the time-out counter
 //starts running.
 pll_reset_asserted  <= `DLY  1'b0;

 if ((TX_QPLL_USED == "TRUE" && qplllock_ris_edge == 1'b1) ||
   (TX_QPLL_USED == "FALSE" && cplllock_ris_edge == 1'b1))
begin
  tx_state  <= `DLY  RELEASE_MMCM_RESET;
  reset_time_out  <= `DLY  1'b1;
end

 if (time_out_2ms == 1'b1)
 begin
    if (retry_counter_int == MAX_RETRIES)
    // If too many retries are performed compared to what is specified in
    // the generic, the counter simply wraps around.
    retry_counter_int <= `DLY  0;
    else
    retry_counter_int <= `DLY  retry_counter_int + 1;
  tx_state           <= `DLY  ASSERT_ALL_RESETS;
 end
end

RELEASE_MMCM_RESET :
begin
 GTTXRESET <= `DLY  1'b0;
 reset_time_out  <= `DLY  1'b0;
 //Release of the MMCM-reset. Waiting for the MMCM to lock.
 MMCM_RESET <= `DLY  1'b0;
 if (mmcm_lock_reclocked == 1'b1)
 begin
```

```
   tx_state <= `DLY  WAIT_RESET_DONE;
   reset_time_out  <= `DLY  1'b1;
  end

  if (time_tlock_max == 1'b1 && mmcm_lock_reclocked == 1'b0 && reset_time_out == 1'b0)
  begin
   if (retry_counter_int == MAX_RETRIES)
     // If too many retries are performed compared to what is specified in
     // the generic, the counter simply wraps around.
     retry_counter_int <= `DLY  0;
   else
     retry_counter_int <= `DLY  retry_counter_int + 1;
   tx_state          <= `DLY  ASSERT_ALL_RESETS;
  end
 end

 WAIT_RESET_DONE :
 begin
  TXUSERRDY <= `DLY  1'b1;
  reset_time_out  <= `DLY  1'b0;
  if (txresetdone_s3 == 1'b1)
  begin
   tx_state     <= `DLY  DO_PHASE_ALIGNMENT;
   reset_time_out  <= `DLY  1'b1;
  end

  if (time_out_500us == 1'b1 && reset_time_out == 1'b0)
  begin
   if (retry_counter_int == MAX_RETRIES)
     // If too many retries are performed compared to what is specified in
     // the generic, the counter simply wraps around.
     retry_counter_int <= `DLY  0;
   else
     retry_counter_int <= `DLY  retry_counter_int + 1;
   tx_state          <= `DLY  ASSERT_ALL_RESETS;
  end
 end

 DO_PHASE_ALIGNMENT :
 begin
  //The direct handling of the signals for the Phase Alignment is done outside
  //this state-machine.
  RESET_PHALIGNMENT      <= `DLY  1'b0;
  run_phase_alignment_int <= `DLY  1'b1;
  reset_time_out         <= `DLY  1'b0;

  if (PHALIGNMENT_DONE == 1'b1)
   tx_state       <= `DLY  RESET_FSM_DONE;

  if (time_out_wait_bypass_s3 == 1'b1)
  begin
   if (retry_counter_int == MAX_RETRIES)
     // If too many retries are performed compared to what is specified in
     // the generic, the counter simply wraps around.
     retry_counter_int <= `DLY  0;
   else
     retry_counter_int <= `DLY   retry_counter_int + 1;
   tx_state          <= `DLY  ASSERT_ALL_RESETS;
  end
 end

 RESET_FSM_DONE :
 begin
  reset_time_out       <= `DLY  1'b1;
  tx_fsm_reset_done_int <= `DLY  1'b1;
 end

 default:
   tx_state          <= `DLY  INIT;
```

```
        endcase
      end
    end

endmodule


module gtx8_chan_pll_RX_STARTUP_FSM #
  (
    parameter    EXAMPLE_SIMULATION      = 0,         // Set to 1 for Simulation
    parameter    GT_TYPE                 = "GTX",
    parameter    EQ_MODE                 = "DFE",      //Rx Equalization Mode - Set to DFE or LPM
    parameter    STABLE_CLOCK_PERIOD     = 8,          //Period of the stable clock driving this state-machine, unit is [ns]
    parameter    RETRY_COUNTER_BITWIDTH  = 8,
    parameter    TX_QPLL_USED            = "FALSE",    // the TX and RX Reset FSMs must
    parameter    RX_QPLL_USED            = "FALSE",    // share these two generic values

    parameter     PHASE_ALIGNMENT_MANUAL  = "TRUE"       // Decision if a manual phase-alignment is necessary or the
automatic
                                        // is enough. For single-lane applications the automatic alignment is
                                        // sufficient
  )
  (
    input      wire    STABLE_CLOCK,     //Stable Clock, either a stable clock from the PCB
                              //or reference-clock present at startup.
    input      wire    RXUSERCLK,        //RXUSERCLK as used in the design
    input      wire    SOFT_RESET,       //User Reset, can be pulled any time
    input      wire    QPLLREFCLKLOST,   //QPLL Reference-clock for the GT is lost
    input      wire    CPLLREFCLKLOST,   //CPLL Reference-clock for the GT is lost
    input      wire    QPLLLOCK,         //Lock Detect from the QPLL of the GT
    input      wire    CPLLLOCK,         //Lock Detect from the CPLL of the GT
    input      wire    RXRESETDONE,
    input      wire    MMCM_LOCK,
    input      wire    RECCLK_STABLE,
    input      wire    RECCLK_MONITOR_RESTART,
    input      wire    DATA_VALID,
    input      wire    TXUSERRDY,              //TXUSERRDY from GT
    input      wire    DONT_RESET_ON_DATA_ERROR, //Used to control the Auto-Reset of FSM when Data Error is detected
    output     reg     GTRXRESET = 1'b0,
    output     reg     MMCM_RESET = 1'b1,
    output     reg     QPLL_RESET = 1'b0,        //Reset QPLL (only if RX usese QPLL)
    output     reg     CPLL_RESET = 1'b0,        //Reset CPLL (only if RX usese CPLL)
    output             RX_FSM_RESET_DONE,        //Reset-sequence has sucessfully been finished.
    output     reg     RXUSERRDY = 1'b0,
    output     wire     RUN_PHALIGNMENT,
    input      wire    PHALIGNMENT_DONE,
    output     reg     RESET_PHALIGNMENT = 1'b0,
    output     reg     RXDFEAGCHOLD  = 1'b0,
    output     reg     RXDFELFHOLD  = 1'b0,
    output     reg     RXLPMLFHOLD  = 1'b0,
    output     reg     RXLPMHFHOLD  = 1'b0,
    output     wire     [RETRY_COUNTER_BITWIDTH-1:0] RETRY_COUNTER // Number of
                                                  // Retries it took to get the transceiver up and running
    );


//Interdependencies:
// * Timing depends on the frequency of the stable clock. Hence counters-sizes
//   are calculated at design-time based on the Generics
//
// * if either of the PLLs is reset during TX-startup, it does not need to be reset again by RX
//   => signal which PLL has been reset
// *



  localparam [3:0]
          INIT              = 4'b0000,
          ASSERT_ALL_RESETS    = 4'b0001,
          RELEASE_PLL_RESET    = 4'b0010,
```

```
            VERIFY_RECCLK_STABLE = 4'b0011,
            RELEASE_MMCM_RESET   = 4'b0100,
            WAIT_RESET_DONE      = 4'b0101,
            DO_PHASE_ALIGNMENT   = 4'b0110,
            MONITOR_DATA_VALID   = 4'b0111,
            FSM_DONE             = 4'b1000;

    reg [3:0] rx_state = INIT;

    //This function decides how many clock-cycle need to be waited until
    // a time-out occurs for bypassing the TX-Buffer
    function [12:0] get_max_wait_bypass;
      input manual_mode;
      reg [12:0] max_wait_cnt;
    begin
      if (manual_mode == "TRUE")
        max_wait_cnt = 5000;
      else
        max_wait_cnt = 3100;
      get_max_wait_bypass = max_wait_cnt;
    end
    endfunction

    localparam integer MMCM_LOCK_CNT_MAX = 1024;
    localparam integer STARTUP_DELAY = 500;//AR43482: Transceiver needs to wait for 500 ns after configuration
    localparam integer WAIT_CYCLES = STARTUP_DELAY / STABLE_CLOCK_PERIOD; // Number of Clock-Cycles to wait after
configuration
    localparam integer WAIT_MAX = WAIT_CYCLES + 10;                    // 500 ns plus some additional margin
     localparam integer WAIT_TIMEOUT_2ms   = 2000000 / STABLE_CLOCK_PERIOD;  //2 ms time-out
    localparam integer WAIT_TLOCK_MAX     = 100000 / STABLE_CLOCK_PERIOD;   //100 us time-out
    localparam integer WAIT_TIMEOUT_500us = 500000 / STABLE_CLOCK_PERIOD;   //500 us time-out
    localparam integer WAIT_TIMEOUT_1us   = 1000 / STABLE_CLOCK_PERIOD;     //1 us time-out
    localparam integer WAIT_TIMEOUT_100us = 100000 / STABLE_CLOCK_PERIOD;   //100us time-out
    integer    WAIT_TIME_ADAPT    = (37000000 /6.4)/STABLE_CLOCK_PERIOD;

    reg [7:0] init_wait_count = 0;
    reg       init_wait_done = 1'b0;
    reg       pll_reset_asserted = 1'b0;

    reg       rx_fsm_reset_done_int = 1'b0;
    wire      rx_fsm_reset_done_int_s2;
    reg       rx_fsm_reset_done_int_s3 = 1'b0;

    localparam integer MAX_RETRIES = 2**RETRY_COUNTER_BITWIDTH-1;
    reg [7:0]  retry_counter_int = 0;
    reg [18:0] time_out_counter = 0;
    reg [1:0]  recclk_mon_restart_count = 0 ;
    reg        recclk_mon_count_reset = 0;

    reg        reset_time_out = 1'b0;
    reg        time_out_2ms = 1'b0;  //--\Flags that the various time-out points
    reg        time_tlock_max = 1'b0; //--|have been reached.
    reg        time_out_500us = 1'b0; //--|
    reg        time_out_1us = 1'b0;   //--|
    reg        time_out_100us = 1'b0;  //--/
    reg        check_tlock_max = 1'b0;

    reg [9:0]  mmcm_lock_count = 1'b0;
    reg        mmcm_lock_int = 1'b0;
    wire       mmcm_lock_i;
    reg        mmcm_lock_reclocked = 1'b0;

    reg       run_phase_alignment_int = 1'b0;
    wire      run_phase_alignment_int_s2;
    reg       run_phase_alignment_int_s3 = 1'b0;


    localparam integer MAX_WAIT_BYPASS = 5000;//5000 RXUSRCLK cycles is the max time for Multi Lane designs

    reg [12:0] wait_bypass_count = 0;
```

```
 reg       time_out_wait_bypass = 1'b0;
 wire       time_out_wait_bypass_s2;
 reg       time_out_wait_bypass_s3 = 1'b0;

 wire       refclk_lost;

 wire      rxresetdone_s2;
 reg      rxresetdone_s3 = 1'b0;

 wire      data_valid_sync;

 wire      cplllock_sync;
 wire      qplllock_sync;
 reg      cplllock_ris_edge = 1'b0;
 reg      qplllock_ris_edge = 1'b0;
 reg      cplllock_prev;
 reg      qplllock_prev;


 integer    adapt_count = 0;
 reg       time_out_adapt = 1'b0;
 reg       adapt_count_reset = 1'b0;

 //Alias section, signals used within this module mapped to output ports:
 assign    RETRY_COUNTER     = retry_counter_int;
 assign    RUN_PHALIGNMENT   = run_phase_alignment_int;
 assign    RX_FSM_RESET_DONE = rx_fsm_reset_done_int;

 always @(posedge STABLE_CLOCK)
 begin
    // The counter starts running when configuration has finished and
    // the clock is stable. When its maximum count-value has been reached,
    // the 500 ns from Answer Record 43482 have been passed.
    if (init_wait_count == WAIT_MAX)
       init_wait_done <= `DLY  1'b1;
    else
      init_wait_count <= `DLY  init_wait_count + 1;
 end



 always @(posedge STABLE_CLOCK)
 begin
   //This counter monitors, how many retries the CDR Lock Detection
   //runs. If during startup too many retries are necessary, the whole
   //initialisation-process of the transceivers gets restarted.
    if (recclk_mon_count_reset == 1)
      recclk_mon_restart_count <= `DLY  0;
    else if (RECCLK_MONITOR_RESTART == 1)
    begin
      if (recclk_mon_restart_count == 3)
        recclk_mon_restart_count <= `DLY  0;
      else
        recclk_mon_restart_count <= `DLY  recclk_mon_restart_count + 1;
    end
 end

generate
 if(EXAMPLE_SIMULATION == 1)
 begin
   always @(posedge STABLE_CLOCK)
   begin
      time_out_adapt <= `DLY  1'b1;
   end
 end

 else
 begin
  always @(posedge STABLE_CLOCK)
  begin
```

```
    if (adapt_count_reset == 1'b1)
    begin
      adapt_count    <= `DLY  0;
      time_out_adapt <= `DLY  1'b0;
    end
    else
    begin
      if (adapt_count == WAIT_TIME_ADAPT -1)
        time_out_adapt <= `DLY  1'b1;
      else
        adapt_count <= `DLY  adapt_count + 1;
    end
  end

 end
endgenerate

  always @(posedge STABLE_CLOCK)
  begin
    // One common large counter for generating three time-out signals.
    // Intermediate time-outs are derived from calculated values, based
    // on the period of the provided clock.
    if (reset_time_out == 1)
    begin
      time_out_counter  <= `DLY  0;
      time_out_2ms      <= `DLY  1'b0;
      time_tlock_max    <= `DLY  1'b0;
      time_out_500us    <= `DLY  1'b0;
      time_out_1us      <= `DLY  1'b0;
      time_out_100us    <= `DLY  1'b0;
    end
    else
    begin
      if (time_out_counter == WAIT_TIMEOUT_2ms)
        time_out_2ms <= `DLY  1'b1;
      else
        time_out_counter <= `DLY  time_out_counter + 1;

      if (time_out_counter > WAIT_TLOCK_MAX && check_tlock_max == 1)
      begin
        time_tlock_max <= `DLY  1'b1;
      end

      if (time_out_counter == WAIT_TIMEOUT_500us)
      begin
        time_out_500us <= `DLY  1'b1;
      end

      if (time_out_counter == WAIT_TIMEOUT_1us)
      begin
        time_out_1us <= `DLY  1'b1;
      end

      if (time_out_counter == WAIT_TIMEOUT_100us)
      begin
        time_out_100us <= `DLY  1'b1;
      end

    end
  end

  always @(posedge STABLE_CLOCK)
  begin
    //The lock-signal from the MMCM is not immediately used but
    //enabling a counter. Only when the counter hits its maximum,
    //the MMCM is considered as "really" locked.
    //The counter avoids that the FSM already starts on only a
    //coarse lock of the MMCM (=toggling of the LOCK-signal).
    if (mmcm_lock_i == 1'b0)
    begin
```

```
      mmcm_lock_count <= `DLY  0;
      mmcm_lock_reclocked   <= `DLY  1'b0;
    end
    else
    begin
      if (mmcm_lock_count < MMCM_LOCK_CNT_MAX - 1)
        mmcm_lock_count <= `DLY  mmcm_lock_count + 1;
      else
        mmcm_lock_reclocked <= `DLY  1'b1;
    end
 end


 //Clock Domain Crossing

gtx8_chan_pll_sync_block sync_run_phase_alignment_int
      (
        .clk            (RXUSERCLK),
        .data_in        (run_phase_alignment_int),
        .data_out       (run_phase_alignment_int_s2)
      );

gtx8_chan_pll_sync_block sync_rx_fsm_reset_done_int
      (
        .clk            (RXUSERCLK),
        .data_in        (rx_fsm_reset_done_int),
        .data_out       (rx_fsm_reset_done_int_s2)
      );


 always @(posedge RXUSERCLK)
 begin
    run_phase_alignment_int_s3 <= `DLY run_phase_alignment_int_s2;

    rx_fsm_reset_done_int_s3   <= `DLY rx_fsm_reset_done_int_s2;
 end

gtx8_chan_pll_sync_block sync_time_out_wait_bypass
      (
        .clk            (STABLE_CLOCK),
        .data_in        (time_out_wait_bypass),
        .data_out       (time_out_wait_bypass_s2)
      );

gtx8_chan_pll_sync_block sync_RXRESETDONE
      (
        .clk            (STABLE_CLOCK),
        .data_in        (RXRESETDONE),
        .data_out       (rxresetdone_s2)
      );

gtx8_chan_pll_sync_block sync_mmcm_lock_reclocked
      (
        .clk            (STABLE_CLOCK),
        .data_in        (MMCM_LOCK),
        .data_out       (mmcm_lock_i)
      );

gtx8_chan_pll_sync_block sync_data_valid
      (
        .clk            (STABLE_CLOCK),
        .data_in        (DATA_VALID),
        .data_out       (data_valid_sync)
      );


gtx8_chan_pll_sync_block sync_cplllock
      (
        .clk            (STABLE_CLOCK),
        .data_in        (CPLLLOCK),
```

```
        .data_out       (cplllock_sync)
      );

gtx8_chan_pll_sync_block sync_qplllock
      (
        .clk            (STABLE_CLOCK),
        .data_in        (QPLLLOCK),
        .data_out       (qplllock_sync)
      );


 always @(posedge STABLE_CLOCK)
 begin
   time_out_wait_bypass_s3   <= `DLY time_out_wait_bypass_s2;

   rxresetdone_s3           <= `DLY rxresetdone_s2;
   cplllock_prev            <= `DLY cplllock_sync;
   qplllock_prev            <= `DLY qplllock_sync;
 end

 always @(posedge STABLE_CLOCK)
 begin
   if(SOFT_RESET == 1'b1)
     cplllock_ris_edge <= 1'b0;
   else if((cplllock_prev == 1'b0) && (cplllock_sync == 1'b1))
     cplllock_ris_edge <= 1'b1;
   else if(rx_state == ASSERT_ALL_RESETS || rx_state == RELEASE_PLL_RESET)
     cplllock_ris_edge <= cplllock_ris_edge;
   else
     cplllock_ris_edge <= 1'b0;
 end

 always @(posedge STABLE_CLOCK)
 begin
   if((qplllock_prev == 1'b0) && (qplllock_sync == 1'b1))
     qplllock_ris_edge <= 1'b1;
   else if(rx_state == ASSERT_ALL_RESETS || rx_state == RELEASE_PLL_RESET)
     qplllock_ris_edge <= qplllock_ris_edge;
   else
     qplllock_ris_edge <= 1'b0;
 end



 always @(posedge RXUSERCLK)
 begin
    if (run_phase_alignment_int_s3 == 1'b0)
    begin
      wait_bypass_count     <= `DLY  0;
      time_out_wait_bypass  <= `DLY  1'b0;
    end
    else if ((run_phase_alignment_int_s3 == 1'b1) && (rx_fsm_reset_done_int_s3 == 1'b0))
    begin
      if (wait_bypass_count == MAX_WAIT_BYPASS - 1)
        time_out_wait_bypass <= `DLY  1'b1;
      else
        wait_bypass_count <= `DLY  wait_bypass_count + 1;
    end
 end

  assign refclk_lost = ( RX_QPLL_USED == "TRUE"  && QPLLREFCLKLOST == 1'b1) ? 1'b1 :
                  ( RX_QPLL_USED == "FALSE" && CPLLREFCLKLOST == 1'b1) ? 1'b1 : 1'b0;


 //FSM for resetting the GTX/GTH/GTP in the 7-series.
 //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 //
 // Following steps are performed:
 // 1) After configuration wait for approximately 500 ns as specified in
 //    answer-record 43482
```

```
// 2) Assert all resets on the GT and on an MMCM potentially connected.
//    After that wait until a reference-clock has been detected.
// 3) Release the reset to the GT and wait until the GT-PLL has locked.
// 4) Release the MMCM-reset and wait until the MMCM has signalled lock.
//    Also get info from the TX-side which PLL has been reset.
// 5) Wait for the RESET_DONE-signal from the GT.
// 6) Signal to start the phase-alignment procedure and wait for it to
//    finish.
// 7) Reset-sequence has successfully run through. Signal this to the
//    rest of the design by asserting RX_FSM_RESET_DONE.

always @(posedge STABLE_CLOCK)
begin
   if (SOFT_RESET == 1'b1 || (rx_state != INIT && rx_state != ASSERT_ALL_RESETS && refclk_lost == 1'b1))
   begin
    rx_state              <= `DLY  INIT;
    RXUSERRDY             <= `DLY  1'b0;
    GTRXRESET             <= `DLY  1'b0;
    MMCM_RESET            <= `DLY  1'b1;
    rx_fsm_reset_done_int  <= `DLY  1'b0;
    QPLL_RESET            <= `DLY  1'b0;
    CPLL_RESET            <= `DLY  1'b0;
    pll_reset_asserted     <= `DLY  1'b0;
    reset_time_out         <= `DLY  1'b1;
    retry_counter_int      <= `DLY  0;
    run_phase_alignment_int <= `DLY  1'b0;
    check_tlock_max        <= `DLY  1'b0;
    RESET_PHALIGNMENT      <= `DLY  1'b1;
    recclk_mon_count_reset  <= `DLY  1'b1;
    adapt_count_reset      <= `DLY  1'b1;
    RXDFEAGCHOLD           <= `DLY  1'b0;
    RXDFELFHOLD            <= `DLY  1'b0;
    RXLPMLFHOLD            <= `DLY  1'b0;
    RXLPMHFHOLD            <= `DLY  1'b0;
   end
   else
   begin

    case (rx_state)
      INIT :
      begin
       //Initial state after configuration. This state will be left after
       //approx. 500 ns and not be re-entered.
       if (init_wait_done == 1'b1)
         rx_state  <= `DLY  ASSERT_ALL_RESETS;
      end

      ASSERT_ALL_RESETS :
      begin
       //This is the state into which the FSM will always jump back if any
       //time-outs will occur.
       //The number of retries is reported on the output RETRY_COUNTER. In
       //case the transceiver never comes up for some reason, this machine
       //will still continue its best and rerun until the FPGA is turned off
       //or the transceivers come up correctly.
        if (RX_QPLL_USED == "TRUE" && TX_QPLL_USED == "FALSE")
        begin
         if (pll_reset_asserted == 1'b0)
         begin
           QPLL_RESET         <= `DLY  1'b1;
           pll_reset_asserted  <= `DLY  1'b1;
         end
         else
           QPLL_RESET         <= `DLY  1'b0;
        end
       else if (RX_QPLL_USED == "FALSE" && TX_QPLL_USED)
       begin
         if (pll_reset_asserted == 1'b0)
         begin
           CPLL_RESET <= `DLY  1'b1;
```

```
            pll_reset_asserted  <= `DLY  1'b1;
          end
          else
            CPLL_RESET         <= `DLY  1'b0;
        end
        RXUSERRDY              <= `DLY  1'b0;
        GTRXRESET              <= `DLY  1'b1;
        MMCM_RESET             <= `DLY  1'b1;
        run_phase_alignment_int <= `DLY  1'b0;
        RESET_PHALIGNMENT      <= `DLY  1'b1;
        check_tlock_max        <= `DLY  1'b0;
        recclk_mon_count_reset  <= `DLY  1'b1;
        adapt_count_reset      <= `DLY  1'b1;

        if ((RX_QPLL_USED == "TRUE" && TX_QPLL_USED == "FALSE" && QPLLREFCLKLOST == 1'b0 && pll_reset_asserted) ||
          (RX_QPLL_USED == "FALSE"&& TX_QPLL_USED == "TRUE"  && CPLLREFCLKLOST == 1'b0 && pll_reset_asserted) ||
          (RX_QPLL_USED == "TRUE" && TX_QPLL_USED == "TRUE"  && QPLLREFCLKLOST == 1'b0 ) ||
          (RX_QPLL_USED == "FALSE"&& TX_QPLL_USED == "FALSE" && CPLLREFCLKLOST == 1'b0 )
          )
        begin
          rx_state           <= `DLY  RELEASE_PLL_RESET;
          reset_time_out      <= `DLY  1'b1;
        end
        end

        RELEASE_PLL_RESET :
        begin
        //PLL-Reset of the GTX gets released and the time-out counter
        //starts running.
        pll_reset_asserted  <= `DLY  1'b0;
        reset_time_out     <= `DLY  1'b0;

        if ((RX_QPLL_USED == "TRUE"  && TX_QPLL_USED == "FALSE" && qplllock_ris_edge == 1'b1) ||
          (RX_QPLL_USED == "FALSE" && TX_QPLL_USED == "TRUE"  && cplllock_ris_edge == 1'b1))
        begin
          rx_state             <= `DLY  VERIFY_RECCLK_STABLE;
          reset_time_out        <= `DLY  1'b1;
          recclk_mon_count_reset  <= `DLY  1'b0;
          adapt_count_reset      <= `DLY  1'b0;
        end
        else if ((RX_QPLL_USED == "TRUE"  && qplllock_sync == 1'b1) ||
              (RX_QPLL_USED == "FALSE" && cplllock_sync == 1'b1))
        begin
          rx_state             <= `DLY  VERIFY_RECCLK_STABLE;
          reset_time_out        <= `DLY  1'b1;
          recclk_mon_count_reset  <= `DLY  1'b0;
          adapt_count_reset      <= `DLY  1'b0;
        end

        if (time_out_2ms == 1'b1)
        begin
          if (retry_counter_int == MAX_RETRIES)
            // If too many retries are performed compared to what is specified in
            // the generic, the counter simply wraps around.
            retry_counter_int <= `DLY  0;
          else
          begin
            retry_counter_int <= `DLY  retry_counter_int + 1;
          end
          rx_state          <= `DLY  ASSERT_ALL_RESETS;
        end
        end

        VERIFY_RECCLK_STABLE :
        begin
        //reset_time_out  <= `DLY  '0';
        //Time-out counter is not released in this state as here the FSM
        //does not wait for a certain period of time but checks on the number
```

```
 //of retries in the CDR PPM detector.
 GTRXRESET <= `DLY  1'b0;
 if (RECCLK_STABLE == 1'b1)
 begin
   rx_state       <= `DLY  RELEASE_MMCM_RESET;
   reset_time_out  <= `DLY  1'b1;
 end

 if (recclk_mon_restart_count == 2)
 begin
   //If two retries are performed in the CDR "Lock" (=CDR PPM-detector)
   //the whole initialisation-sequence gets restarted.
   if (retry_counter_int == MAX_RETRIES)
     // If too many retries are performed compared to what is specified in
     // the generic, the counter simply wraps around.
     retry_counter_int <= `DLY  0;
   else
   begin
     retry_counter_int <= `DLY  retry_counter_int + 1;
   end
   rx_state          <= `DLY  ASSERT_ALL_RESETS;
 end
end

RELEASE_MMCM_RESET :
begin
 //Release of the MMCM-reset. Waiting for the MMCM to lock.
 reset_time_out  <= `DLY  1'b0;
 check_tlock_max <= `DLY  1'b1;

 MMCM_RESET <= `DLY  1'b0;
 if (mmcm_lock_reclocked == 1'b1)
 begin
   rx_state <= `DLY  WAIT_RESET_DONE;
   reset_time_out  <= `DLY  1'b1;
 end

 if (time_tlock_max == 1'b1 && reset_time_out  == 1'b0)
 begin
   if (retry_counter_int == MAX_RETRIES)
     // If too many retries are performed compared to what is specified in
     // the generic, the counter simply wraps around.
     retry_counter_int <= `DLY  0;
   else
   begin
     retry_counter_int <= `DLY  retry_counter_int + 1;
   end
   rx_state          <= `DLY  ASSERT_ALL_RESETS;
 end
end

WAIT_RESET_DONE :
begin
//When TXOUTCLK is the source for RXUSRCLK, RXUSERRDY depends on TXUSERRDY
//If RXOUTCLK is the source for RXUSRCLK, TXUSERRDY can be tied to '1'

 if(TXUSERRDY)
    RXUSERRDY    <= `DLY  1'b1;

 reset_time_out  <= `DLY  1'b0;
 if (rxresetdone_s3 == 1'b1)
 begin
   rx_state       <= `DLY  DO_PHASE_ALIGNMENT;
   reset_time_out  <= `DLY  1'b1;
 end

 if (time_out_2ms == 1'b1 && reset_time_out  == 1'b0)
 begin
   if (retry_counter_int == MAX_RETRIES)
     // If too many retries are performed compared to what is specified in
```

```
            // the generic, the counter simply wraps around.
            retry_counter_int <= `DLY  0;
          else
          begin
            retry_counter_int <= `DLY  retry_counter_int + 1;
          end
          rx_state          <= `DLY  ASSERT_ALL_RESETS;
        end
      end

      DO_PHASE_ALIGNMENT :
      begin
       //The direct handling of the signals for the Phase Alignment is done outside
       //this state-machine.
       RESET_PHALIGNMENT      <= `DLY  1'b0;
       run_phase_alignment_int <= `DLY  1'b1;
       reset_time_out         <= `DLY  1'b0;

       if (PHALIGNMENT_DONE == 1'b1)
       begin
         rx_state       <= `DLY  MONITOR_DATA_VALID;
         reset_time_out  <= `DLY  1'b1;
       end

       if (time_out_wait_bypass_s3 == 1'b1)
       begin
         if (retry_counter_int == MAX_RETRIES)
           // If too many retries are performed compared to what is specified in
           // the generic, the counter simply wraps around.
           retry_counter_int <= `DLY  0;
         else
         begin
           retry_counter_int <= `DLY   retry_counter_int + 1;
         end
         rx_state          <= `DLY  ASSERT_ALL_RESETS;
       end
      end

      MONITOR_DATA_VALID :
      begin
       reset_time_out  <= `DLY  1'b0;

       if (data_valid_sync == 1'b0 && time_out_100us == 1'b1 && DONT_RESET_ON_DATA_ERROR == 1'b0 &&
reset_time_out  == 1'b0)
       begin
         rx_state           <= `DLY  ASSERT_ALL_RESETS;
         rx_fsm_reset_done_int <= `DLY  1'b0;
       end
       else if (data_valid_sync == 1'b1)
       begin
         rx_state           <= `DLY  FSM_DONE;
         rx_fsm_reset_done_int <= `DLY  1'b0;
         reset_time_out       <= `DLY  1'b1;
       end

      end

      FSM_DONE :
      begin
       reset_time_out       <= `DLY  1'b0;

       if (data_valid_sync == 1'b0)
       begin
          rx_fsm_reset_done_int <= `DLY  1'b0;
          reset_time_out       <= `DLY  1'b1;
          rx_state            <= `DLY  MONITOR_DATA_VALID;
       end
       else if(time_out_1us == 1'b1 && reset_time_out  == 1'b0)
          rx_fsm_reset_done_int <= `DLY  1'b1;
```

```
          if(time_out_adapt)
          begin
            if(GT_TYPE == "GTX"  && EQ_MODE == "DFE")
            begin
              RXDFEAGCHOLD  <= `DLY 1'b1;
              RXDFELFHOLD   <= `DLY 1'b1;
            end
            else
            begin
              RXDFEAGCHOLD  <= `DLY 1'b0;
              RXDFELFHOLD   <= `DLY 1'b0;
              RXLPMHFHOLD   <= `DLY 1'b0;
              RXLPMLFHOLD   <= `DLY 1'b0;
            end
          end


        end

        default:
          rx_state              <= `DLY  INIT;

      endcase
    end
  end

endmodule
```

REFERENCES

[1] G. E. Moore, “Progress in digital integrated electronics,” in Electron Devices Meeting, 1975 International, vol. 21, pp. 11–13, IEEE, 1975.

[2] International Technology Roadmap for Semiconductors 2009 Edition - Test and Test Equipment, The International Technology Roadmap for Semiconductors, 2009.

[3] "PCI Express 4.0 evolution to 16GT/s, twice the throughput of PCI Express 3.0 technology" (press release). PCI-SIG. 2011-11-29. Retrieved 2012-12-07.

[4] A.M. Majid, D.C. Keezer, “Multi-Function Multi-GHz ATE Extension Using State-Of-The-Art FPGAs,” Proc. of the IEEE Intl. Test Conf. (ITC), 2011.

[5] D.C. Keezer, C. Gray, A. Majid, D. Minier, P. Ducharme, “A Development Platform and Electronic Modules for Automated Test up to 20Gbps,” Proc. of the IEEE Intl. Test Conf. (ITC), Paper 14.3, Austin, Texas, November 2009.

[6] “7 Series FPGAs Overview,” Xilinx, Released 2014-2-18.

[7] “7 Series FPGAs GTX/GTH Transceivers User Guide,” Xilinx, Released 2014-2-11.

[8] Anon.,”about Teradyne – A brief History,” http://www.teradyne.com/corp/history.html

[9] Bierman., H. “VLSI test gear keeps path with chip advances,” Electronics, April 19, 1984.

[10] Furukawa, Y., Rajsuman, R., “New trends drive ATE open architecture,” Semiconductor International, Issue 8, July 2005

[11] Shahriari, N., ”Mission impossible? Open architecture ATE,” Proceedings, IEEE International Test Conference. *(ITC),* 2002.

[12] Verigy V93000Series High-Speed I/O test Solution – Solution Overview, Verigy Ltd., California 2009

[13] Evans, A.C., “The new ATE: Protocol aware,” IEEE International Test Conference. (ITC), pp.1-2, 21-26, October 2007.

[14] Tim Erjavec, White Paper, "Introducing the Xilinx Targeted Design Platform: Fulfilling the Programmable Imperative." February 2, 2009. Retrieved February 2, 2009

[15] Kuon, I.; Rose, J. (2006). "Measuring the gap between FPGAs and ASICs". Proceedings of the internation symposium on Field programmable gate arrays - FPGA'06. p. 21.

[16] “History of FPGA” http://filebox.vt.edu/users/tmagin/history.htm

[17] Clive Maxfield, Programmable Logic DesignLine, "Xilinx unveil revolutionary 65nm FPGA architecture: the Virtex-5 family. May 15, 2006. Retrieved February 5, 2009.

[18] "Altera and Xilinx Report: The Battle Continues". Seeking Alpha. July 17, 2008. Retrieved November 13, 2013.

[19] "Documentation: Stratix IV Devices". Altera.com. 2008-06-11. Retrieved 2013-05-01.

[20] McConnel, Toni. EETimes. "ESC - Xilinx All Programmable System on a Chip combines best of serial and parallel processing." April 28, 2010. Retrieved February 14, 2011.

[21] Nass, Rich, EETimes. "Xilinx puts ARM core into its FPGAs." April 27, 2010. Retrieved February 14, 2011.

[22] Atmel 8-bit AVR Microcontroller Datasheet, http://www.atmel.com/Images/Atmel-2586-AVR-8-bit-Microcontroller-ATtiny25-ATtiny45-ATtiny85_Datasheet.pdf

[23] Actel.com, http://www.microsemi.com/products/fpga-soc/fpga-and-soc

[24] Iniewski, Krzysztof, et al. "*SERDES Technology for Gigabit I/O Communications in Storage Area Networking*." *IWSOC*. 2004.

[25] Zwitter, Deborah, et al. "Competitiveness and Technical Challenges of Low Cost Wirebond Packaging for High Speed SerDes Applications in ASICs." Electronic Components and Technology Conference, 2007. ECTC'07. Proceedings. 57th. IEEE, 2007.

[26] Moreira, Jose, Marc Moessinger, Koji Sasaki, and Takayuki Nakamura. "Driver Sharing Challenges for DDR4 High-Volume Testing with ATE." International Test Conference (ITC), 2012 IEEE International, 1-10: IEEE, 2012.

[27] David.C. Keezer, Te-Hui Chen, Carl E. Gray "Multi-Gigahertz Test Signal Synthesis with Timing-on-the-Fly," International Mixed-Signals, Sensors and Systems Test Workshop. (IM3STW), 2012.

[28] D.C. Keezer, T.H. Chen, C.E. Gray "Multi-Gigahertz Arbitrary Timing Generator and Data Pattern Serializer/Formatter," Proc. of the IEEE International. Test Conference. (ITC), 2012.

[29] Hyunjin Kim and Jacob A Abraham, "A Built-in Self-Test Scheme for DDR Memory Output Timing Test and Measurement," IEEE 30th VLSI Test Symposium (VTS), p.7-12, 2012.

[30] Paolo Bernardi, Michelangelo Grosso, Matteo Sonza Reorda, and Y Zhang, "A Programmable BIST for DRAM Testing and Diagnosis," IEEE International Test Conference (ITC), p.1-10, 2010.

[31] "JEDEC" (2010-12-01). "204-Pin DDR3 SDRAM SO-DIMM Specification"

[32] By 'annihilator' (2010-08-18). "DDR4 memory in Works, Will reach 4.266GHz". wccftech.com. Retrieved 2011-04-29.

[33] Dekker, Rob, Frans Beenker, and Loek Thijssen. "Realistic built-in self-test for static RAMs." Design & Test of Computers, IEEE 6.1 (1989): 26-34.

[34] Abramovici, Miron, Melvin A. Breuer, and Arthur D. Friedman. "Digital systems testing and testable design." Design for Testability (1990).

[35] Vermeulen, Bart, Camelia Hora, Bram Kruseman, Erik Jan Marinissen, and Robert Van Rijsinge. "Trends in testing integrated circuits." International Test Conference (ITC), 2012 IEEE International, pp. 688-697. IEEE, 2004.

[36] "Datasheet of SY89296U: 2.5V/3.3V 1.5MHz Precision LVPECL Programmable Delay with Fine Tune Control." Micrel Corp. http://www.micrel.com/index.php/component/joodb/article/78/2.html

[37] "Datasheet of NBSG53A: SiGe 2.5 V / 3.3 V Selectable Differential Clock / Data D Flip-Flop / Clock Divider with Reset and OLS" On Semiconductor. http://www.onsemi.com/PowerSolutions/product.do?id=NBSG53A

[38] "Datasheet of HMC745: 13 Gbps Fast Rise Time XOR / XNOR Gate with Programmable Output Voltage " Analog Devices Inc. http://www.analog.com/en/products/high-speed-logic/hmc745.html

[39] "Spartan-6 Family Overview" Xilinx Inc., page1-3 http://www.analog.com/en/products/high-speed-logic/hmc745.html

[40] "Spartan-6 FPGA SelectIO Resources User Guide," page 24, Xilinx Inc. http://www.xilinx.com/support/documentation/user_guides/ug381.pdf

[41] "ISE Design Suite 14: Release Notes, Installation, and Licensing," Xilinx Inc. http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/irn.pdf

[42] "RO4000® Series High Frequency Circuit Materials," Rogers Corp. https://www.rogerscorp.com/documents/726/acs/RO4000-LaminatesData-sheet.pdf

[43] "Microstrip" http://www.microwaves101.com/encyclopedias/microstrip

[44] "Stripline" http://www.microwaves101.com/encyclopedias/stripline

[45] "High-Speed Layout Guidelines," page3-7, Texas Instruments, http://www.ti.com/lit/an/scaa082/scaa082.pdf

[46] "7 Series FPGAs Overview," Xilinx Inc., page1-6, http://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf

[47] "7 Series FPGAs SelectIO Resources," Xilinx Inc., http://www.xilinx.com/support/documentation/user_guides/ug471_7Series_SelectIO.pdf

[48] "7 Series FPGAs GTX/GTH Transceivers," Xilinx Inc., http://www.xilinx.com/support/documentation/user_guides/ug476_7Series_Transceivers.pdf

[49] T. Moon, H.W. Choi, D.C. Keezer, A. Chatterjee, "Multi-channel testing architecture for high-speed eye-diagram using pin electronics and subsampling monobit reconstruction algorithms," Proc. the IEEE VLSI Test Symposium (VTS), 2014

[50] Jun Kohno, Tatsuro Akiyama, Dai Kato and Makoto Imamura, "A High Linearity Compact Timing Vernier for CMOS Timing Generator," Proceedings IEEE International Test Conference, 2010, paper 1.1, pp. 1-8.

[51] Masakatsu Suda, Kazuhiro Yamamoto, Toshiyuki Okayasu, Shusuke Kantake, Satoshi Sudou, and Daisuke Watanabe, "CMOS High-Speed, High-Precision Timing Generator for 4.266-Gbps Memory Test System," Proceedings IEEE International Test Conference, 2005, paper 34.2, pp.1-9.

[52] Arkin, Brian. "Lowering the Cost of Test with a Scalable ATE Custom Processor and Timing IC Containing 400 High-Linearity Timing Verniers." Proceedings IEEE International Test Conference, 2005, paper 34.4, pp. 1-6.

[53] Ting-Yuan Wang, Shih-Min Lin, and Hen-Wai Tsao, "Multiple Channel Programmable Timing Generators with Single Cyclic Delay Line," IEEE Trans. on Instrumentation and Measurement, VOL. 53, NO. 4, August 2004.

[54] A. Dabrowski, P. Pawlowski, K. Setecki, "Programmable Clock Signal Generators for Sc Structures," Mixed Design of Integrated Circuits and System, MIXDES 2006, Proceedings of the International Conference, pp.369-373.

[55] Jiaqi Li, Zhe Zheng, Min Liu, Siliang Wu, "Large Dynamic Range Accurate Digitally Programmable Delay Line with 250-Ps Resolution," International Conference on Signal Processing, 2006.

[56] Sun Zhaolin, Li Nan, Wang Yinan, Yin Qinghong, Xu Xin, Guo Jing, Liu Haijun, Xu Hui, "High Resolution Programmable Digital Delay Generator Design and Realization," International Conference on Intelligent System Design and Engineering Application, 2010, pp.813-816.

[57] Sadok Aouini, Kun Chuai, and Gordon W. Roberts. "A Low-Cost Ate Phase Signal Generation Technique for Test Applications." Proceedings IEEE International Test Conference, 2010, paper 1.4, pp. 1-10.

[58] J. Moreira, M. Moessinger, K. Sasaki, and T. Nakamura, "Driver sharing challenges for DDR4 high-volume testing with ATE Proceedings IEEE International Test Conference, 2012.

[59] H. Kim and J.A. Abraham, "A built-in self-test scheme for DDR memory output timing test and measurement," IEEE 30th VLSI Test Symposium (VTS), p.7-12, 2012.

[60] P. Bernardi, M. Grosso, M.S. Reorda, and Y. Zhang, "A programmable BIST for DRAM testing and diagnosis," Proceedings IEEE International Test Conference, 2010.

[61] R.L. Ladbury, M.D. Berg, E.P. Wilcox, K.A. LaBel, H.S. Kim, A.M. Phan, and C.M. Seidleck, "Use of commercial FPGA-based evaluation boards for single-event

testing of DDR2 and DDR3 SDRAMS," IEEE Trans. on Nuclear Science, Vol.60, No.6, p.4457-4463, 2013.

[62] I. Aleksejev, A. Jutman, S. Devadze, S. Odintsov, and T. Wenzel, "FPGA-based synthetic instrumentation for board test," Proceedings IEEE International Test Conference, 2012.

[63] J. Ferry, "FPGA-based universal embedded digital instrument," Proceedings IEEE International Test Conference, 2013.

[64] S.H. Lee, S. Cho, K.J. Song, E.J. Byun, S.H. Joo, S.D. Suh, K. Ha, S.J. Oh, and W. Lee, "A serial optical link based memory test system for high-speed and multi-parallel test." J. of Lightwave Technology, 28, no. 1, p.104-110, 2010.

[65] S. Kojima, Y. Arai, T. Fujibe, T. Ataka, A. Ono, K. Sawada, D. Watanabe, "8Gbps CMOS pin electronics hardware macro with simultaneous bi-directional capability," Proceedings IEEE International Test Conference, 2012.

[66] D.C. Keezer, C. Gray, A. Majid, D. Minier, P. Ducharme, "A development platform and electronic modules for automated test up to 20Gbps," Proceedings IEEE International Test Conference, 2009.

[67] A.M. Majid, D.C. Keezer, "Multi-function multi-GHz ATE extension using state-of-the-art FPGAs," Proceedings IEEE International Test Conference, 2011.

[68] D.C. Keezer, T.H. Chen, C.E. Gray "Multi-gigahertz arbitrary timing generator and data pattern serializer/formatter," Proceedings IEEE International Test Conference, 2012.

[69] "Datasheet of SY89297U: 2.5/3.3V, 3.2Gbps Precision CML Dual-Channel Programmable Delay." Micrel Corp. http://www.micrel.com/index.php/component/joodb/article/78/3.html

[70] "Datasheet of SY58626L: DC-to-6.4Gbps Backplane Transmit Buffer." Micrel Corp. http://www.micrel.com/_PDF/HBW/sy58626l.pdf

[71] "Datasheet of HMC674: 10GHz Latched Comparator, " Analog Devices Inc. http://www.analog.com/en/products/high-speed-logic/hmc674lc3c.html

[72] "Datasheet of HMC874: 20Gbps Latched Comparator, " Analog Devices Inc. http://www.analog.com/en/products/clock-and-timing/clock-generation-distribution/hmc874.html

[73] "Datasheet of GRF312/GRF332: SMT DPDT Non-Latching Electromechanical Relay, " Teledyne Technologies Inc. http://www.teledynerelays.com/pdf/electromechanical/GRF312_GRF332_Datasheet.pdf

[74] "Datasheet of LTC2656: Octal 16-/12-Bit Rail-to-Rail DACs, " Linear Technologies Inc. http://cds.linear.com/docs/en/datasheet/2656fa.pdf

[75] "Datasheet of LT3080: Adjustable1.1A Single Resistor Low Dropout Regulator, " Linear Technologies Inc. http://cds.linear.com/docs/en/datasheet/3080fc.pdf

[76] "SPI Tutorial," http://www.corelis.com/education/SPI_Tutorial.htm

[77] "Datasheet of HMC847: 36 Gbps, 4:1 Mux SMT with Duty Cycle Control & Programmable Output Voltage, " Analog Devices Inc. http://www.analog.com/en/products/high-speed-logic/logic-mux-de-mux/hmc847.html

[78] "Datasheet of HMC848: 45 Gbps 1:4 Demux SMT with Programmable Output Voltage, " Analog Devices Inc. http://www.analog.com/en/products/high-speed-logic/logic-mux-de-mux/hmc848.html

[79] "Kintex-7 FPGAs Data Sheet: DC and AC Switching Characteristics," Xilinx Inc., http://www.xilinx.com/support/documentation/data_sheets/ds182_Kintex_7_Data_Sheet.pdf

[80] "UltraScale Architecture and Product Overview," Xilinx Inc., http://www.xilinx.com/support/documentation/data_sheets/ds890-ultrascale-overview.pdf

[81] "Virtex UltraScale+ FPGA Data Sheet:DC and AC Switching Characteristics," Xilinx Inc., http://www.xilinx.com/support/documentation/data_sheets/ds923-virtex-ultrascale-plus.pdf

[82] "UltraScale Architecture SelectIO Resources," Xilinx Inc., http://www.xilinx.com/support/documentation/user_guides/ug571-ultrascale-selectio.pdf

[83] "UltraScale Architecture GTH Transceivers," Xilinx Inc., http://www.xilinx.com/support/documentation/user_guides/ug576-ultrascale-gth-transceivers.pdf

[84] "UltraScale Architecture GTY Transceivers," Xilinx Inc., http://www.xilinx.com/support/documentation/user_guides/ug578-ultrascale-gty-transceivers.pdf

[85] "High-speed interconnects that support 28+ Gbps requirements," Samtec Inc., https://www.samtec.com/connectors/high-speed-board-to-board/28gbps/28-gbps

VITA

TE-HUI CHEN

Te-Hui was born June 21st 1984 in Chia-Yi, Taiwan. He received a B.S. in Electrical Engineering and a B.A. in Economics from National Chung Cheng University, Chia-Yi County, Taiwan in 2007 before coming to Georgia Tech to pursue a doctorate in Electrical and Computer Engineering.

In the spring of 2011, Te-Hui joined the High-Speed Digital Test Lab at Georgia Tech under the guidance of Dr. David Keezer in order to pursue the Ph.D. in electrical and computer engineering. His interests include high-speed testing, FPGA-based design and development of high-performance and low-cost digital testing system. He has over 6 years of experience in designing High-speed digital test system. In the summer of 2016, Te-Hui received the Ph.D. degree in electrical and computer engineering